SIG32C SYSTEM USERS MANUAL

(revision 1.3)
September, 1995

Signalogic, Inc.
9704 Skillman #111
Dallas, TX 75243
tel: 214-343-0069
fax: 214-343-0163

TABLE OF CONTENTS

1. INTRODUCTION 1-1
1.1 Read Me First!. 1-2
1.1.1 Conventions Used in this Manual 1-3
1.2 General Capabilities 1-3
1.3 Specifications 1-5
2. INSTALLATION 2-1
2.1 Board Switches and Jumpers 2-2
2.1.1 I/O Base Address Select. 2-2
2.1.2 PC DMA Channel Select 2-2
2.1.3 PC Interrupt Select 2-3
2.1.4 External Connector Power 2-4
3. THE PROCESSOR 3-1
3.1 Memory Map 3-2
3.2 Serial Port 3-5
3.3 Interrupts 3-6
3.4 Host Port Interface 3-7
4. CONTROL/STATUS REGISTER 4-1
5. QWBUS 5-1
5.1 Interprocessor Communications. 5-1
5.2 Parallel Digital I/O 5-2
5.3 Synchronizing Interboard Sampling. 5-3
6. ANALOG I/O INTERFACE 6-1
6.1 Sampling Rates 6-2
6.2 Analog Inputs and Outputs 6-3
6.2.1 Programmable Input Gain 6-5
6.2.2 Input Voltage Range 6-6
6.2.3 Input Impedance 6-6
6.3 CS4216 Information. 6-7
7. SOFTWARE 7-1
7.1 Hypersignal Software 7-3
7.1.1 Hypersignal DSP32C Source Code Interface 7-9
7.1.1.1 Source Code License Agreement 7-13
7.1.1.2 Source Code Overview 7-13
7.1.1.3 Source Code and Support File List 7-14
7.1.2.4 Source Code Modification Process 7-17
7.1.2.5 Source Code Initialization Examples 7-23

TABLE OF CONTENTS

7.2 DSPower Software 7-3
7.3 Debuggers / Monitor Programs 7-27
7.4 DOS C and Windows C/C++ Libraries 7-28
Appendix A: Glossary A-1
Appendix B: Block-Diagram Schematics B-1
Appendix C: CS4216 Data Sheets. C-1

Sig32C System Users Manual

this page intentionally left blank

1. INTRODUCTION

Welcome to the Sig32C System. This system combines floating-point Digital Signal Processing (DSP) power with multiple analog input and output channels on personal computer (PC) plug-in boards, in order to solve signal processing problems.

The boards are plugged into PCs, and high-level PC software is then used to both control the boards and download executable code native to the DSP processor on the boards. Multiple signals from the outside world can then be connected to the endplate connectors on the boards for processing to extract meaningful information.

Signals of all types and physical characteristics can be digitized, including sound, pressure, vibration, seismic, physiological, etc. as long as a transducer that converts the signal to electrical voltage is used. In cases where the natural sampling rates of the signal are less than the minimum sampling rate range of the boards, software “frequency zoom” methods can be used to shift sampled data down to the desired bandwidth. In most cases, the processing power of one or more Sig32C boards is enough to allow continuous, real-time processing of incoming signals.

The Sig32C System is designed for both single-board use and multiple boards to be connected together, allowing up to six (6) processors to work and communicate together, and up to 48 (forty-eight) analog I/O channels to be synchronized. Currently, three basic types of boards are available:

- the Sig32C-8, which contains the DSP32C 32-bit floating-point processor, memory, interboard communication bus, parallel digital I/O, and eight (8) analog I/O channels
- the Sig32C, which is the Sig32C-8 with no analog I/O subsection
- the Sig32C-P, which contains processor, memory, and a “prototyping” area intended for user-defined I/O circuitry or modules. No interboard communication or pre-defined parallel digital I/O is available. The prototyping area provides access to the DB37 endplate connector, as well as DSP32C serial port and the onboard parallel bus normally used to access external peripheral devices.

Note also that customized versions of all of the various Sig32C boards may be available, depending on the application and volume of boards required. See section 1.2 below, General Capabilities, for more detailed information on what can be done with the Sig32C System.

All Sig32C System boards can be operated at one of three processor speeds: 50, 64, and 80 MHz. All boards also support one of three SRAM sizes: 32k x 32, 128k x 32, and 512k x 32.

All boards are fully supported by Hypersignal DOS software, and by DSPower Windows software if Hypersignal is used as a “DSP Engine”. See section 7.1, Hypersignal and DSPower, for more information about these two high-level packages. Additionally, several utilities and support tools are provided to enable user-defined source code development. See chapter 7, SOFTWARE, for more information about supporting software in general.

1.1 Read Me First!

This manual is for both the Sig32C-8 processor board with eight (8) analog I/O channels,the Sig32C processor board with no analog I/O, and the Sig32C-P prototyping board with user-defined analog or other I/O subsection. Since all Sig32C boards contain the general-purpose, programmable DSP32C device, a DSP32C program must be downloaded to to the board before it can do useful work. In order to put Sig32C System boards to work, you must do one of the following:

- use Hypersignal software, which include several instruments, displays, and math/DSP functions that cause DSP32C code programs to be downloaded to the board at various times; Hypersignal contains built-in drivers to control and transfer programs and data to and from the board. Three programs are available: Hypersignal-Macro, Hypersignal-Macro EX, and Hypersignal-Acoustic.

- use DSPower-Block Diagram software, which provides a graphical block diagram, display and instrument user-interface, and allows use of Hypersignal software as a “DSP Engine” where Sig32C boards are concerned. DSPower-Block Diagram provides both simulation and interactive control modes of operation.
- use DSPower-HWLib software, which is a Windows DLL that offers a library of calls to interface user-defined C/C++ programs to Sig32C boards; low-level functions include board initialization and control, memory transfer, DSP program (COFF file) download, and semaphore management; high-level functions include continuous acquisition/generation to/fromhard disk, continuous function generation, stimulus & response measurement, and more
- modify or add to existing Hypersignal DSP32C source code to change DSP32C operation, but continue to use Hypersignal and/or DSPower-HWLib to download programs and control board operation (see chapter 7, SOFTWARE). The Hypersignal DSP32C Source Code Interface software package contains run-time math and DSP functions, and analog I/O drivers for the Sig32C-8.
- use the CAC D3BUG utility, which contains DOS C library functions to allow access to Sig32C boards from host PC programs; D3BUG also contains a source-level debugger which enables debug of arbitrary assembly language or C programs running on the DSP32C.

Note that for the methods listed above except stand-alone use of Hypersignal and/or DSPower-Block Diagram software, you will most likely need to do some programming: either for the DSP32C processor, or for the host. Any time you need to write or change DSP32C source code, you will need at least the AT&T DSP32C assembler and linker. If real-time processing constraints allow it—or the size of the project is large—the AT&T DSP32C ‘C’ Compiler is also recommended. In addition, the AT&T DSP32C Software Simulator, Family Application Library, and Hypersignal DSP32C Source Code Interface are recommended tools for any level of DSP32C source code development. If host programming is required and DSPower-HWLib software is being used, then a Windows C/C++ compiler such as Borland C/C++ v3.1, Microsoft C/C++ v1.5, or WATCOM C/C++ v10.0 is recommended. If the D3BUG C library is being used, then a DOS C compiler such as Microsoft Quick C, Borland C/C++ v2.0, or other is suggested.

Understanding Sig32C System hardware is essential to using it effectively. We recommend that you read and understand this manual, as well as the AT&T DSP32C Information Manual. **Please read this manual.** We realize that spending time reading documentation is not everyone’s strong suite, so we have tried to be concise, and present information in a well-organized manner. If you have questions not answered in this manual, please contact Signalogic technical support.

1.1.1 Conventions Used in this Manual

Some of the abbreviations, symbols, and other conventions used in this manual are listed below:

Numbers starting with “0x” or ending with “H” are hexadecimal values.

Text surrounded by a single quote, as in ‘xxx’, is intended to show a software entry. Examples include data fields, dialog boxes, setup menus, etc.

“V” is used to indicate Volts, and “dB” to indicate a log gain value, where dB = 20*log(gain).

All binary values, typically found in tables showing how to program hardware setup registers, are read as msb-to-lsb from left-to-right.

NOTE: In the case of unusually important information, we use this “Note” format to bring it to your attention.

1.2 General Capabilities

The Sig32C-8 board combines the power of the AT&T 32-bit floating-point DSP32C processor, and up to 512k x 32 of SRAM (external data and program memory for the DSP32C), with eight (8) channels of analog input, ***and*** eight (8) channels of output. Each analog I/O channel has a maximum sampling frequency of 48 kHz, making this board suitable for

applications such as digital audio processing, vibration analysis, and various types of multichannel signal analysis, instrumentation, and waveform synthesis.

The Sig32C board is the Sig32C-8 board without the analog I/O subsection, but including the interboard communication bus and external parallel digital I/O. Because of the extensive interprocessor communications subsystem designed into the Sig32C System, multiboard combinations such as one Sig32C-8 and one or more Sig32Cs for extra processing power are cost-effective and straightforward to use. The Sig32C-P board is a “prototyping” version of the Sig32C, with a large prototyping area for user-defined circuitry or modules replacing board real-estate normally used for analog I/O and interboard communication bus.

The AT&T DSP32C processor is a fast 32-bit floating-point device with an instruction cycle time as low as 50 nsec. Three different speeds of Sig32C boards are available: 81, 62.5, and 50 nsec. All instructions complete in one cycle, providing performance up to 20 MIPs and 40 MFLOPS. The MFLOP rate is twice because of the general format of each DSP32C instruction:

$$a = b + c * d$$

where a can be a memory location or floating-point accumulator, and b, c, and d can be combinations of memory locations and accumulators. Note that up to four memory accesses may also be performed in the same instruction. To achieve the maximum number of operations in a single instruction, the DSP32C uses a highly pipelined architecture. The result is a level of performance, and compactness of program code size, that rank among the best DSP devices on the market. See chapter 4, THE PROCESSOR, for more detailed information about the DSP32C.

There are three (3) memory (SRAM) sizes available for Sig32C System boards: 32k x 32, 128k x 32, 512k x 32. At least the 128k x 32 amount is strongly recommended, unless it is certain that the application does not require additional memory, and/or the extra cost savings is important. The availability of up to 128k x 32 of SRAM allows Sig32C System boards to meet a wide range of program and data memory requirements. Since DSP program sizes are typically less than 32k x 32, a total memory amount of 128k x 32 leaves a considerable amount of SRAM available for data buffers. Note that Sig32C System boards are not “field upgradable”; i.e. a 32k x 32 board can only be converted to a 128k x 32 board by return to the factory. Note that for the 512k x 32 memory size, monolithic 512k x 8 SRAM devices are used, which are currently significantly more expensive. It is estimated that by the end of 1996, 512k x 8 SRAM device costs will be “normalized”.

The Sig32C System has been designed as a complete solution to multichannel acquisition. Each channel has programmable gain, differential input, programmable input coupling, and uses state-of-the-art sigma-delta technology, which allows software-selectable sampling rates, superb anti-alias input filters and reconstruction output filters that track with the sampling rate, and nearly perfect linear phase. All inputs and outputs sample simultaneously, and are DMA’ed directly into DSP memory for optimal DSP32C software efficiency. Analog I/O sampling rates can be generated by onboard oscillators under software control, or provided by external clock signals (from other Sig32C boards or user-defined sources).

The parallel multi-master “QWbus” allows multiple boards to communicate and transfer data blocks at high-speed, in order to avoid conflict with the host PC I/O bus. With the Sig32C System, up to six (6) boards can be interconnected together using the QWbus, which operates at 19.7 Mbyte/second; furthermore, the QWbus can be used to synchronize analog I/O sampling between multiple boards, allowing up to 48 channels to sample simultaneously.

The QWbus also allows parallel digital I/O to the outside world, by using 16-bit memory-mapped DSP32C addresses.

1.3 Specifications

Physical

Height	4.5"
Length	7.5"
Analog I/O Connector (Sig32C-8 only)	37-pin female “D-sub”, endplate (external)

Parallel Digital I/O Connector	40-pin ribbon cable, male header (internal)
Interprocessor Communications Connector	40-pin ribbon cable, male header (internal)

Typical Analog I/O Specifications (Sig32C-8)

Instrumentation Amplifiers	Burr-Brown PGA204AP (±5V supply, eight per board)
A/D and D/A Converters	Crystal Semiconductor CS4216-KL stereo CODECs (four per board)
Input Voltage Range	±1.4 Volts (Gain = 0 dB) ±0.105 mV (Gain = 82.5 dB)
Maximum Input Voltage	±40 Volts to ground
Input Impedance	10 kOhm || 12 pF (can be altered; see section 6.2.3)
Output Voltage Range	±2.98 Volts (Atten = 0 dB) ±223 mV (Atten = -22.5 dB)
Maximum Output Current	Outputs are short-circuit protected
Output Impedance	0.01 Ohm
Output Load Resistance	600 Ohm minimum
Analog Input Filter	80 dB stopband rejection, ±0.2 dB passband ripple
Analog Output Filter	74 dB stopband rejection, ±0.1 dB passband ripple
Input Dynamic Range	82 dB (typical)
Output Dyanmic Range	82 dB (typical)
Input Signal-to-(Noise+ Distortion) Ratio	80 dB @ f_{in} = 1 kHz, gain = 1
Output Signal-to-(Noise+ Distortion) Ratio	72 dB @ f_{out} = 1 kHz, gain = 1

Maximum Power Consumption (Sig32C-8)

@ +5V	1.9 Amp
@ ±12V	80 mA

Maximum Power Consumption (Sig32C)

@ +5V	1.5 Amp
@ ±12V	not used

2. INSTALLATION

Installation of Sig32C System boards is not hard. In the case of use with Hypersignal or DSPower software, you will probably have to do nothing but plug the board into your PC (following the sequence below). In all cases, most of the board's operation and its resources are software-selectable, so there are few jumpers and switches to be concerned with during installation.

The basic installation sequence is as follows:

1) Turn off and ***unplug*** your computer. Remember to "park" your hard disk drive first if your computer requires this. If you fail to unplug your computer before installation of the board, and you damage the board or yourself because the computer was still plugged in (or turned on—don't believe it doesn't happen), then you were stupid or temporarily insane or off in space somewhere and Signalogic is not responsible. Needless to say, all and any board warranties are void if damage occurs in this way.

2) Set the board's base I/O address switches as explained in section 2.1.1 below. The factory setting is 290H; if you are reasonably sure that there are no other plug-in boards in your computer using this I/O address, then save section 2.1.1 for later, and proceed using the factory setting.

3) Install the board into your computer. You will have to remove the cover from the computer first. You may also have to remove a "dummy" backplate from the motherboad bus slot in which you choose to plug the board. Make sure that you plug the board into an "AT" bus slot, which is one that has two connectors. Some machines offer a mix of AT and "XT" motherboard bus slots; XT slots have only one connector. The Sig32C System boards will not work correctly in an XT slot.

4) Reinstall the cover to your computer, plug your computer back in, and turn it back on. If your computer doesn't boot up, or beeps, or any other behavior occurs to which you are not accustomed, then ***turn the computer off again immediately.*** In such a case, it may be possible that you did not seat the board fully into its connectors, or that the board's base I/O address is conflicting with another board.

5) To verify the board is operating correctly, run Hypersignal software or DSPower-HWLib demo programs (digital oscilloscope, tape recorder, etc.). In the DSPower-HWLib demo programs, a dialog box is displayed that shows a list of supported hardware and allows Sig32C System baords to be selected from the list. For Hypersignal operation, see Hypersignal Users Manual section 2.5.2, Initializing the System Configuration Menu, for simple instructions on selecting the SigC31-4 board device driver inside Hypersignal.

2.1 Board Switches and Jumpers

There are a number of jumpers and switches situated immediately above the ISA bus ("AT bus") connector on the board. These determine the interface to the PC, including the board's base I/O address, DMA channel (if used), and PC interrupt (if used). Correct setting of the base I/O address switches is vital to proper board operation; switch settings should be verified prior to installation of the board. For most cases of operation, including software control by Hypersignal or user-written programs using the C Library functions, the DMA channel and interrupt jumpers can be left in an open (unconnected) position.

2.1.1 Base I/O Address Select

There are two "hex" switches that determine the base I/O address of the board in the 4k byte addressing range of the PC ISA bus. The base I/O address should be within a range of 140H to 340H (where the 'H' indicates hex, or hexadecimal). Other addresses are possible both lower and higher, but risk interference with typical PC peripherals, such as hard disk controllers, mouse and communications serial ports, printer ports, etc. The default, factory configuration base I/O address is 290H. The most significant digit of the base I/O address (i.e. '2' for the default address) is set by SW1 (switch 1), and the middle digit is set by SW2. The least significant digit is implied, and is always zero.

> NOTE: The SW2 setting also affects the board's interprocessor QWbus address in a multiple board system. In the QWbus communications scheme, each board must have a unique address to which to respond. See chapter 5, QWBUS, for more information.

2.1.2 PC DMA Channel Select

The PC DMA channel, should you use software that requires it, can be set via jumpers LK1 and LK2. Connecting the two pins marked "A" (printed on the board) with a shorting jumper selects DMA channel 7, "B" selects channel 6, and "C" selects channel 5. Both LK1 and LK2 must be set exactly the same, because LK1 sets the DMA request signal (DREQ7 through DREQ5) to the host PC's DMA controller, and LK2 directs the DMA acknowledge signal (DACK7 through DACK5) to the controller.

If no DMA is used, LK1 and LK2 should be left open (unconnected).

> NOTE: Hypersignal software, DSPower-HWLib DLL, and the QW32CLIB C library functions do not use a PC DMA channel. If you write host PC software that uses a PC DMA channel, then you must remember to program the host PC's DMA controller to initiate and control the data transfer process, and also to set the DSP32C host port for 16-bit transfers. See section 3.4 below, Host Port Interface, for more information.

2.1.3 PC Interrupt Select

The DSP32C processor can initiate a host PC interrupt if needed. LK3 determines which PC interrupt will be used by connecting the DSP32C's PIF line to one of three (3) possible host PC interrupts. Connecting the two LK3 pins marked "A" (printed on the board) with a shorting jumper selects IRQ12, "B" selects IRQ11, and "C" selects IRQ10.

If no host PC interrupt is used, LK3 should be left open (unconnected).

> NOTE: Hypersignal software, DSPower-HWLib DLL, and the QW32CLIB C library functions do not use a PC interrupt. If you write host PC software that uses a PC interrupt, then you must remember to provide a host software interrupt routine to service the board's interrupt when it occurs. A PC interrupt can be useful if the host PC software must be constructed in such a way as to not require it to poll for completion of data processing ("next data buffer") by the DSP32C. However, note that within DOS and Windows 3.x environments, interrupt response times are notoriously slow, and minimum latency times are nearly impossible to predict.

2.1.4 External Connector Power

Jumper LK4 can be used to allow power (+5V) to be supplied to pin 19 on the 37-pin endplate connector on the board. LK5 can be used to allow power (+5V) to be supplied to pin 40 on the 40-pin QWbus connector on the board. These options are intended to allow external circuitry connected to Sig32C System boards to derive its power from the board if necessary. The factory default leaves these jumpers open to prevent accidental shorts to power when connecting other pins on these connectors.

3. THE PROCESSOR

Sig32C System boards contain an AT&T DSP32C 32-bit floating-point DSP chip operating at either 50, 64, or 80 MHz clock rate (note: version 5.0 and earlier Sig32C boards may contain only two oscillators, in which case only a processor clock rate of 49.152 MHz was available). The 80 MHz rate provides performance of 40 MFLOPs (million floating-point operations per second; instruction cycle time of 50 nsec).

This device has become widely used in DSP-based products as a cost-effective and reliable processor for real-time signals. It has proven popular for a number of reasons, including:

- its “C-like” assembly language
- the power of single instructions, each of which provide for an accumulate and multiply, and up to 4 memory accesses (3 reads and one write) per instruction cycle; because of this format, the DSP32C provides about the same performance for a given function as other DSP chips that offer instruction cycle times up to 25% faster; in addition, the DSP32C usually requires less code to do the same job!
- throughput gained by its pipelined architecture
- easy-to-use host port that allows full access and control by external devices and computers
- simple but effective serial port, onchip DMA, and external interrupt combination

Programming the DSP32C is made straightforward by its assembly language, which bears similarity to ‘C’. A typical instruction might take the form:

[*r1++ =] a0 = a0 + *r2++ * *r3++r15

where the contents of memory location pointed to by r3 are multiplied by contents of memory location pointed to by r2 and the result added to floating-point accumulator a0. The result is place in a0, and optionally written to memory location pointed to by r1. All three memory pointers are post-updated in the above example: r1 and r2 by one 32-bit word amount, and r3 by the amount contained in “increment register” r15. Any combination of the floating point accumlators can be used.

Other basic features of the DSP32C include:

- 32-bit floating-point operation with 24-bit mantissa and 8-bit exponent; instructions to convert between IEEE format
- 8-bit, 16-bit, and 24-bit integer operations (both register and memory)
- 22 general-purpose integer registers (“r” registers), which can be used for various integer operations and as memory pointers in floating-point operations, and a subset of which can also be used as “increment registers” in floating-point operations (see above example)
- 4 floating-point accumulators (and 4 more shadow registers that automatically switch into the foreground during interrupts). Accumlators actually are 40 bits in length, which provides eight “guard” bits to maintain precision in successive accumulator-based instructions, before data is again written out to memory (at which time it is rounded to 32 bits of precision)
- two external interrupt sources, plus serial port receive and transmit interrupts (internal)
- host port (for online access by external devices or computers)
- 32-bit word serial port, with onchip DMA between serial port data and memory

3.1 Memory Map

The table below shows the DSP32C processor memory map on Sig32C System boards:

Mem Area	Addrress Range (in bytes)	Access	Size (byte)	Comments
INT 2	0x000000 0x0007FF	Internal RAM2[3]	2k	zero wait-state, DSP32C memory mode 5
INT 0	0x000800 0x000FFF	Internal RAM0[3]	2k	zero wait-state, DSP32C memory mode 5
EXT A	0x001000 0x020FFF	External RAM (Area A)	128k	zero wait-state; standard configuation of 32k x 32 SRAM installed
	0x021000 0x080FFF	External RAM (Area A), continued	384k	zero wait-state; expanded memory area if 128k x 32 SRAM option is installed
	0x081000 0x5FFFFF	External RAM (Area A), continued	5628k	remainder of EXT A; contains memory images of installed memory
EXT B[1]	0x600000 0x7FFFFF	External RAM (Area B)	2048k	if external RAM accessed here, has same wait-state as other Area B (1, 2, or 3 wait)[1]
	0x800008	QWbus Addr 1	2	QWbus addresses[2] are determined by three LSBs of middle digit of board's host PC I/O base address setting (see section 2.1.1); for example, base I/O address of 290H selects QWbus address 1, 2A0H selects QWbus address 2, etc.
	0x800010	QWbus Addr 2	2	
	0x800020	QWbus Addr 3	2	
	0x800040	QWbus Addr 4	2	
	0x800080	QWbus Addr 5	2	
	0x800100	QWbus Addr 6	2	
	0xA00000 0xA0007F	QWbus Parallel Digital I/O	256	128 16-bit locations to address external I/O peripherals (image repeats to 0xBFFFFF)
	0xC00000	QWbus FIFO Read	2	read data sent by other boards on QWbus (read-only)
	0xE00003	Control/Status Register	1	write to Control register, read from status register
INT 1	0xFFF800 0xFFFFFF	Internal RAM1[3]	2k	zero wait-state, DSP32C memory mode 5

Notes:

[1]Ext B area is programmable to 1, 2, or 3 wait-states via PCW register. See section 2.15.2 in AT&T DSP32C Information Manual.

[2]A write to QWbus address will transfer 16-bit data to the FIFO on the addressed board. A read will cause INTREQ2 to occur on the addressed board. More than one board can be addressed simultaneously by combining (logical OR) QWbus addresses into one address.

[3]Internal RAM locations reflect the fact that Sig32C System board DSP32Cs operate in memory mode 5.

Both internal and external memory can be accessed by the DSP32C as bytes, 16-bit integers, or 32-bit words. Floating-point values and instructions (program op codes) are always 32 bits. Integers can be 16-bits or 24-bits. The DSP32C is the only floating-point processor on the market that allows more efficient use of memory by storing some data in a short integer format.

Internal processor memory is organized into three (3) areas, each 512 x 32 in size. These are labeled RAM0, RAM1, and RAM2 in the above memory map, providing 1.5k x 32 of zero wait-state internal RAM. The standard external memory configuration of the Sig32C board is 32k x 32 SRAM in the Ext A area, expandable to 128k x 32. Sig32C System boards

are not “field upgradable”; because memory parts are soldered to the board, and because the 32k parts differ from the 128k parts in form factor, boards must be returned to the factory to be upgraded.

Notes:

1) Upon reset, the DSP32C is set for two (2) wait-states for all memory areas. External A should be reprogammed for zero wait-states, external B for one wait-state. One wait-state is 1/4 instruction cycle, or 20 nsec, and are programmed into the pcw register. See section 7.1.1.5, Source Code Initialization Examples. On Sig32C System boards, 32k x 32 SRAM devices are in 28-pin DIP packages, and 128k x 32 in 32-pin DIP packages.

2) Write accesses to address range 0x800000 - 0x8001FE will write 16-bit data into the FIFO of one or more Sig32C boards connected to the QWbus (see chapter 5, QWBUS). Only the least significant 6 bits of the write address is used to access the QWbus. Each Sig32C board on the QWbus responds to one address line of these 6 bits; which line is determined by the I/O base address of the board (again, see chapter 5). Note that more than one Sig32C board can be addressed simultaneously (broadcast mode). Reads in this address range will cause INTREQ2 to occur on the addressed board(s). Typical programming convention uses INTREQ2 to inform receiving boards when their FIFOs are full and need to be emptied; this allows fast, efficient block data transfers to take place between interconnected boards. Both writes and reads to this address range cause the FS (function select) line to be zero.

3) Accesses in the address range 0xA00000 - 0xA001FE will read or write 16-bit data to peripheral external devices/boards connected to the QWbus. 128 unique addresses are supported. Accesses in this address range cause the FS (function select) line to be one.

4) Accesses to address 0xE00003 cause a byte to be read or written from the Sig32C Control register. The Control register is used for several purposes, including selecting onboard oscillator-generated sampling rates, enabling QWbus FIFOs, synchronizing sampling between multiple boards, and more. See chapter 4, CONTROL/STATUS REGISTER.

3.2 Serial Port

On the Sig32C-8 board, the serial port on the DSP32C chip is devoted to input and output of digitized analog I/O data. On Sig32C boards (no analog I/O subsection), the serial port is available for user-defined oepration.

When serial port operation on Sig32C System boards is configured for automatic DMA into processor memory, all eight (8) input and output channels (or other user-defined I/O data) are transferred to and from specified memory locations during each sample period without code intervention. At the end of each sample period, a “frame” interrupt is provided that allows DSP32C code to move input samples to other memory areas (buffers) and prepare output samples for subsequent output.

For Sig32C-8 boards, analog I/O data is transferred between memory and the four (4) CS4216 devices (each device is stereo, or two channels each; see below) via the serial port. The CS4216 devices automatically output digitized A/D format converter data and accept D/A converter data in serial format. Serial data is transmitted and received by the CS4216s most-significant-bit first. The double-buffering used by the DSP32C serial port results in data being stored in the following order:

Receive Data Offset[1]		Transmit Data Offset[1]	
0x00	chan 0	0x00	chan 2[2]
0x04	chan 1	0x04	chan 3
0x08	chan 2	0x08	chan 4
0x0C	chan 3	0x0C	chan 5
0x10	chan 4	0X10	chan 6
0x14	chan 5	0X14	chan 7
0x18	chan 6	0X18	chan 0
0x1C	chan 7	0x1C	chan 1

Receive Data Offset[1]	Transmit Data Offset[1]
Notes: [1]At run-time, these offset values must be added to the base address values inside the r21 and r22 registers. The DSP32C uses r21 and r22 as serial port DMA input and output pointers, respectively. See section 7.1.1.5, Source Code Initialization Examples, for a source code example of initializing r21 and r22. [2]D/A channel values are shifted because of the double-buffering in the DSP32C serial port output buffer.	

Each CS4216 converter is a stereo device, containing two input and two output channels. Thus, there are four CS4216s on each Sig32C-8 board, with each one occupying a 64-bit subframe with a 256-bit serial data frame (per sample). The overall serial data transmission rate is thus fixed at 256 times the sample rate clock, which is also the master frequency required by the sigma-delta converters. Serial port frame and sync signals are generated to transfer 8 x 32 bit input words and 8 x 32 bit output words for each sample period, punctuated by an INTREQ1 to inform the DSP32C that the analog I/O datastream needs processing. Note that 32-bit words are transmitted, but analog I/O samples are only 16 bits. In each 32-bit word, the upper half contains sample data, and the lower half either input status (from the CS4216s) or output control information (to the CS4216s).

The IOC register on the DSP32C is used to configure the serial port for correct operation. See section 7.1.1.5, Source Code Initialization Examples, for a source code example of programming the IOC register.

3.3 Interrupts

The DSP32C can service two external interrupts, which are requested by input pins INTREQ1 and INTREQ2. Interrupt acknowledge signals, IACK1 and IACK2, are asserted for the duration of the interrupt service routine associated with the INTREQn signal.

On Sig32C-8 boards, INTREQ1 is dedicated as the “frame” interrupt for analog I/O input. This interrupt occurs once each sample period, and indicates that all eight (8) input channels have been transferred to DSP32C memory from the serial port (via DMA), and all 8 output channels have been transferred from memory to the serial port output buffer (also via DMA).

On all Sig32C System boards, INTREQ2 is dedicated to QWbus communications. Reading from a memory-mapped QWbus interboard location causes INTREQ2 to occur on the addressed board(s). This technique is normally used to signal to other boards that their FIFOs contain data that needs to be processed.

The DSP32C has only one level of interrupt priority. Interrupts are latched and will always be serviced unless masked by the PCW register or occur again during the service of their own initial request. Priority is as follows, from highest to lowest: INTREQ1, PDF (host port parallel data register full), PDE (host port parallel data register empty), IBF (serial input buffer full), OBE (serial output buffer empty), and INTREQ2.

3.4 Host Port Interface

This section contains a brief overview of the host port inside the DSP32C processor, and how it is interfaced to the host PC. For detailed information, please refer to chapter 6 in the AT&T DSP32C Information Manual.

The host port on the DSP32C allows for program downloading, real-time, continuous data transfer, and external control over the DSP32C. The host port is mapped into the host PC’s I/O address space, occupying 16 bytes starting at the board’s I/O base address (see section 2.1.1 above).

All program and data transfers between host PC software and the host port are performed using the DSP32C’s internal “cycle-stealing” mechanism. This offers two advantages:

- the host port can be accessed transparently to a program currently executing on the DSP32C
- all DSP32C resources, including on-chip RAM, can be accessed via the host port since it is the DSP32C itself that makes the actual physical read or write

There are three ways of mapping the host port: two 8-bit modes and a 16-bit mode, depending on bits 1 and 9 in the PCR (processor control register, see table below). On reset the host port always returns to 8-bit mode, for reasons of compatibility with previous DSP32 chips. Hypersignal software always puts the host port on Sig32C boards into 16-bit mode after reset and initialization; this is the most efficient and recommended way to use Sig32C boards.

In the Sig32C implementation, the order of the DSP32C host port registers does not follow the AT&T order defined in chapter 6 of the AT&T DSP32C Information Manual. Instead, the order has been changed to make access of the host port more efficient, and make host PC DMA possible:

Input Address A0 A1 A2 A3	8-bit mode	16-bit mode	PAB[3..0] P3 P2 P1 P0
0 0 0 0	PDR2 low byte	PDR2 word (address = 0)	1 1 0 0
0 0 0 1	PDR2 high byte		1 1 0 1
0 0 1 0	PDR low byte	PDR word (address = 2)	0 0 1 0
0 0 1 1	PDR high byte		0 0 1 1
0 1 0 0	PAR low byte	PAR word (address = 4)	0 0 0 0
0 1 0 1	PAR high byte		0 0 0 1
0 1 1 0	PARE byte	PARE byte	1 0 1 1
0 1 1 1	... not used ...		X X X X
1 0 0 0	PCR low byte	PCR word (address = 8)	0 1 1 1
1 0 0 1	PCR high byte		1 0 1 0
1 0 1 0	PIR low byte	PIR word (address = 0xA)	1 0 0 0
1 0 1 1	PIR high byte		1 0 0 1
1 1 0 0	EMR low byte	EMR word (address = 0xC)	0 1 0 0
1 1 0 1	EMR high byte		0 1 0 1
1 1 1 0	ESR byte	ESR byte	0 1 1 0
1 1 1 1	... not used ...		X X X X

Notes:

1) The DSP32C always powers up running, regardless of what happens to be in its program memory at the time. Writing 0 to pcr[0] halts the DSP. A reset then occurs when a 1 is written to pcr[1]. After reset, the DSP32C host port is in 8-bit mode. From a user-written 'C' program, the following sequence could be used to ensure correct initialization after reset (host PC I/O base address of 290H is assumed):

```
outpw(0x298, 0x0002); // halt DSP, assert rest, select 8-bit mode (and assume that some
                      //  how the chip may be in 16-bit mode)
outp(0x298, 0x02); // repeat above sequence, knowing that 8-bit mode is ensured
outp(0x299, 0x02); // select 16-bit mode
```

The above sequence is used by the QwAssertReset() routine in the C Library.

2) 8-bit mode is set by writing pcr[1] to 1; 16-bit mode by writing pcr[9] to 1.

3) Host PC DMA transfers are invoked by DACKn (see section 2.1.2, PC DMA Channel Select) low. The address is not relevant (because the DSP32C host port uses autoincrement addressing) and 8-bit transfers are not supported.

3. THE PROCESSOR

this page intentionally left blank

4. CONTROL/STATUS REGISTER

The board's Control/Status register is shown in the table below. The Control register is accessed by writing a byte value to DSP32C memory address 0xE00003. All eight bits can be read by reading a byte value from the same location.

The Control register is used to enable and control operation of the analog I/O channels and the QWbus. Specifically, it contains 1 to 4-bit subfields that, depending on how they are set, perform the following functions:

- designate sample rate clock source as Provider mode (onboard oscillators provide clock, which is also output to other boards), Dependent mode (external clock provided by other boards or other source), Independent mode (onboard oscillators provide clock, but output drivers to other boards are disabled), or Reset mode, where the ADC and DAC channels are held in power down mode
- set onboard oscillator-select values (to generate analog I/O sampling rate clocks)
- reset onboard CS4216 A/D and D/A converters (CODECs)
- reset onboard QWbus FIFOs used for interprocessor (interboard) communications

See section 6.1, Sampling Rates, for a table containing the possible oscillator-select values, and the analog I/O sampling rates that correspond to these values. Below is a table defining the operation of the Control register bits:

bit 7 not used	bit 6 FIFO Reset	bits 5..4 CODEC Control	bits 3..0 Oscillator-Select
0	0=reset 1=operating	00 = reset CODECs (internal clock is disabled) 11 = Independent mode (internal clock, QWbus SYNC and SCLK outputs disabled)) 10 = Dependent mode (external clock is input) 11 = Provider mode (onboard clock is input, QWbus SYNC and SCLK outputs enabled)	see sampling rate value table

Notes:

1) All bits are cleared to zero at power-up.

2) The reset to CODECs must be at least 500 nsec. Since they are held reset after power-up, this is normally satisfied by enabling the CODECs one time from DSP32C code. However, if resetting the CODECs at some other time, allow at least 10 instruction cycles between asserting and removing reset signal. See section 7.1.1.5, Source Code Initialization Examples, for source code excerpts that wait for calibration to complete before beginning operation.

3) Independent mode is normally used in single-board operation. If multiple boards are connected via the QWbus, any that use non-synchronized sampling rates should use Independent mode. Provider and Dependent modes are used to synchronize sampling rates of multiple boards connected over the QWbus. One board is designated Provider, and provides the serial clock SCLK and frame sync SYNC to the other boards, designated Dependents. Since the QWbus supports up to 6 boards, up to 48 simultaneously sampling channels are possible (up to 48 channels may be synchronized).

4) Bit 6 of the Control register is used to rest the QWbus FIFO, clearing both the Read and Write pointers in the FIFO to zero, in preparation for receiving a block of data from another board, or sending a block of data.

5) The QWbus FIFO reset signal is, like all other bits in the Control register, set to zero at power up. However, to avoid the possibility of the DSP32Cs on more than one board inadvertently attempting to appoint themselves QWbus Master at the same time, which could cause a QWbus signal conflict, the following unique code sequence must first be written to the Control register to enable QWbus output drivers:

0x0A, 0x0A, 0x0C, 0x0C

These four bytes must be written to the Control register in the above sequence to enable QWbus drivers. See section 5.1, Interprocessor Communications, for more information on QWbus data transfers.

Four (4) bits are used to select one of 16 possible onboard sampling rates, generated by a combination of two onboard oscillators. The following sampling rate values are possible:

Sample Rate (kHz)	Control Register bits 3..0
48	0000
44.1	0001
32	0010
29.4	0011
24	0100
22.05	0101
19.2	0110
17.64	0111
16	1000
14.7	1001
12	1010
11.025	1011
9.6	1100
8.82	1101
8	1110
7.35	1111

Notes:

1) Hypersignal software allows continuous entry of sampling rates from 7.35 kHz to 48 kHz in its various instruments and functions, but chooses the sampling rate from the above values that is closest. This is also true for DSPower software operation when Hypersignal is a DSP Engine.

2) For continuous transfer of data to/from hard disk, the maximum “aggregate throughput rate” (i.e. sampling rate x number of channels) will depend on the speed of the host system—especially the I/O bus and hard disk. Continuous aggregate rates above 16 kHz may not be possible.

4. CONTROL/STATUS REGISTER

this page intentionally left blank

5. QWBUS

The parallel bus on Sig32C System boards, called the QWbus, is designed to be connected to other Sig32C boards or external peripheral devices via a 40-pin ribbon cable. The QWbus is intended to perform three functions: allow efficient interprocessor communications and block data transfer between boards, allow parallel digital I/O to the outside world, and allow synchronized analog I/O sampling between interconnected Sig32C-8 boards.

5.1 Interprocessor Communications

When used to interconnect multiple Sig32C System boards, the QWbus allows high-speed transfers of 16-bit data between boards. Up to six (6) boards can be connected together using standard 40-pin ribbon cable between the QWbus connectors located at the top of the boards. Each board has a unique QWbus address that is derived from the setting of its host PC I/O base address (see table below). Any board may write data to any other board, including itself. Data output from a board can be sent to any other board individually, or can be broadcast to multiple boards simultaneously. This is possible because each QWbus address is based on one bit in the QWbus address word, and more than one address can thus be OR'ed together.

Blocks of data are received into onboard 512-byte FIFOs. The sending board can cause an interrupt (INTREQ2) to occur in the DSP32C of the receiving board(s) when a block of data has been written. This is caused by a dummy read of the addressed board(s) after all FIFO writes have been completed. INTREQ2 is cleared by the DSP32C entering the interrupt service routine. Block lengths can be any length—whatever is established by the DSP32C code user-defined programming convention currently in use.

Each board's QWbus address is determined by the current setting of its host PC I/O base address. SW1 and SW2 control the I/O base address value (see section 2.1.1). Below are the possible SW2 settings, and the resulting QWbus addresses:

SW2 Setting	QWbus Address
9	1 (responds if FS = 1, and A1 = 1)
0xA	2 (responds if FS = 1, and A2 = 1)
0xB	3 (responds if FS = 1, and A3 = 1)
0xC	4 (responds if FS = 1, and A4 = 1)
0xD	5 (responds if FS = 1, and A5 = 1)
0xE	6 (responds if FS = 1, and A6 = 1

Notes:

1) To use any of the QWbus interprocessor facilities, one board must be in Master mode. To put a board into Master mode, which enables output from that board's QWbus drivers, a unique code sequence must be first written to the Control register (see chapter 4, CONTROL/STATUS REGISTER).

2) It is completely the responsibility of the programmer to arbitrate data transfers on the QWbus, and establish programming conventions to be adhered to by all Sig32C System boards connected to the QWbus. The board doing the writing must be in Master mode, and boards receiving must be in Slave mode. Boards can be informed of data waiting in their FIFOs by the Master performing a dummy read at the the same (write) address. This causes an interrupt (INTREQ2) to the addressed board(s). There can only be one Master at a time, although series resistors are in place on all QWbus output lines in order to provide protection against short periods of bus conflict.

3) SW2 settings of 1, 2, 3, 4, 5, and 6 produce the same QWbus addresses as 9, A, B, C, D, and E above (i.e. only the 3 lsbs of the address are significant). SW2 settings of 0, 7, 8, or F disable QWbus operation.

5.2 Parallel Digital I/O

Peripheral devices such as other PC plug in boards, other types of analog I/O or digitizing circuitry, trigger or conditional inputs, relay and servo outputs, etc. can also be connected to the QWbus. 128 16-bit locations are mapped into the DSP32C's memory address spaced; access to these locations does not interfere with QWbus interprocessor communication addresses, although programming must ensure that the two types of access do not occur at the same time. Parallel I/O accesses over the QWbus are distinguished from interboard QWbus communications by the FS line (if FS = 0, then the access is interboard, if FS = 1, then the access is external; see section 3.1, Memory Map, for more information). Each of the 128 parallel I/O locations is 16-bits, with a nominal access time of 102 nsec, programmable to 122 or 142 nsec using extra wait-states (for standard Sig32C System board with 49.152 MHz processor clock, each wait-state duration is 1/4 of a processor instruction cycle, or 20 nsec).

External peripheral devices can interrupt Sig32C-8 boards over the QWbus by causing INTREQ2 to occur on a target board. This functionality is shared with QWbus interboard communications, and must be distinguished by the DSP32C software. Peripheral devices that cause interrupts should have some type of status register readable by the DSP32C to indicate that an interrupt is pending; access of this status register by the DSP32C reading it over the QWbus should clear the interrupt. Unlike interboard communications, entering the INTREQ2 service routine won't automatically clear the interrupt, so DSP32C software may distinguish between interboard communications and external device communications by detecting whether the INTREQ2 line is still low inside the service routine.

In all external peripheral parallel I/O transfers the address must be presented first along with the direction (read DIR = 0, write DIR = 1). Some 10 nsec after A0..A6, FS, and DIR are valid, the STRB line is taken low, and held low for some 30 nsec (1 wait state). Data being written must be latched on the rising edge of STRB (setup time = 20 nsec), and data being read must be stable some 20 nsec before this edge. This sequence is required to allow the DSP32C to treat external peripheral devices as memory-mapped I/O locations. Note that parallel I/O accesses take place in the Ext B memory area, which normally runs at 1 wait-state (20 nsec, or 1/4 of DSP32C instruction cycle time).

5.3 Synchronizing Interboard Sampling

The QWbus also allows the analog sampling rate and sampling time of the CS4216 CODECs on up to six (6) boards can be synchronized, allowing up to 48 analog I/O channels to be fully synchronized. One board is designated Provider and other boards Dependents (see chapter 4, CONTROL/STATUS REGISTER). The Provider board drives two signal lines, SCLK (serial clock), and SYNC (frame sync). Dependent boards accept these lines as input and use them to drive their own CS4216 CODECs and serial transmission of digitized sample data between the CODECs and DSP32C processor.

5. QWBUS

this page intentionally left blank

6. ANALOG I/O INTERFACE

NOTE: This chapter applies only to the Sig32C-8 board.

Eight (8) channels of high-quality 16-bit A/D and D/A conversion are provided by four (4) 64x oversampling CS4216 sigma-deta devices. These devices are “stereo CODECs” containing two channels each, and utilize sigma-delta conversion technology, which is based on oversampling and digital filters. Built-in digital filters are used to “downsample”, or decimate oversampled input data to the desired sampling rates, and interpolate or “upsample” output data to the oversampled output rate. Sigma-delta technology offers the advantage of inherent anti-aliasing and reconstruction filters that track with the programmable sampling rate, sharp filter responses, improved signal-to-noise ratios, and nearly perfect linear phase. Disadvantages include delay due to the digital filters, which can become excessive at low sampling rates, and ineffectiveness at very low sampling rates because the oversampled rate no longer offers adequate analog bandlimiting protection. See the Hypersignal Hardware Reference Guide for more information about sigma-delta technology.

Data transfer between the CS4216 CODECs and the DSP32C occur through the DSP32C’s serial port. DMA is used to transfer all 8 input channels and all 8 output channels between DSP32C memory and the serial port transparently to executing DSP32C programs. At the end of each sample period, a “frame” interrupt is generated (on INTREQ1) that indicates all eight channels have been transferred, and the DSP32C should service the interrupt to place the next eight output samples into the DMA area, and transfer the previous eight input samples to a data buffer.

All eight A/D and D/A converters on the Sig32C-8 sample simultaneously. Standard digital audio sample rates can be generated by onboard oscillators or provided externally (see section 6.1, below). When Hypersignal software is used, 16 different sampling rates are software-selectable.

6.1 Sampling Rates

Sample clocks on the Sig32C-8 can be generated from onboard oscillators, or be provided from an external source, including another Sig32C-8 board.

There are three onboard oscillators on the Sig32C-8 board. One is used to provide the DSP32C processor clock, and is not used for analog I/O sampling. The other two oscillators are used to generate 16 possible sampling rates ranging from 7.35 kHz to 48 kHz. Using an integer-divide scheme based on combinations of divide-by-1024 and divide-by-512, the 49.152 MHz (factory default value) oscillator provides 48, 32, 24 kHz, etc. and the 45.158 MHz (factory default value) oscillator provides 44.1, 22.05 kHz, and so on. Note that the sampling rate oscillators may be changed if user-specific sampling rate values are required.

NOTE: Version 5.0 and earlier Sig32C System boards contained only two oscillators. The 49.152 MHz value was used to provide both processor clock and sampling rate input. Version 5.0 and earlier boards were not designed for DSP32C processor speeds above 50 MHz.

The lower four (4) bits of the Control register define the master clock divider N, which results in an effective sampling rate as follows:

Sample rate = master clock / 1024 / (N + 1)

When used with Hypersignal software, the closest sampling rate to what has been entered in instrument menus or displays is chosen. Note that either sampling rate oscillator can be changed to provide custom sampling rates, without affecting the processor clock speed—although in this case, Hypersignal software will not reflect the correct sampling rate without modification. Another option is to use the QWbus external clock input, and set Hypersignal software for external clock operation (‘-EXT’ suffix in System Config menu). See chapter 6, QWBUS, for more information about external clock input.

The table below shows the possible sampling rates for the factory default oscillator values of 49.152 and 45.158 MHz, and the equivalent hex values (divider value N) which must be entered into the Control register:

Sample Rate (kHz)	Control Register bits 3..0
48	0000
44.1	0001
32	0010
29.4	0011
24	0100
22.05	0101
19.2	0110
17.64	0111
16	1000
14.7	1001
12	1010
11.025	1011
9.6	1100
8.82	1101
8	1110
7.35	1111

See chapter 4, CONTROL/STATUS REGISTER, for more detailed information about selecting onboard sampling rate generation or external sampling rate clock, and synchronizing the sampling clocks of multiple Sig32C-8 boards.

NOTES: 1) Hypersignal software allows continuous entry of sampling rates from 7.35 kHz to 48 kHz in its various instruments and functions, but chooses the sampling rate from the values in the table above that is closest. This is also true for DSPower software operation when Hypersignal is a DSP Engine. See, for example, the DSPower-HWLib Digitial Oscilloscope demo; sampling rate entry in this program always “quantizes” to what is allowed on the Sig32C-8 board.

2) For continuous transfer of data to/from hard disk, the maximum “aggregate throughput rate” (i.e. sampling rate x number of channels) will depend on the speed of the host system—especially the I/O bus and hard disk. Continuous aggregate rates above 16 kHz may not be possible. Systems with local bus (e.g. PCI bus) hard disk controllers may be capable of 22.05 kHz or higher for all four (4) channels.

6.2 Analog Inputs and Outputs

See figure 6-1 below for the pin-out of the analog I/O connector located on the endplate of Sig32C-8 boards.

6. ANALOG I/O INTERFACE

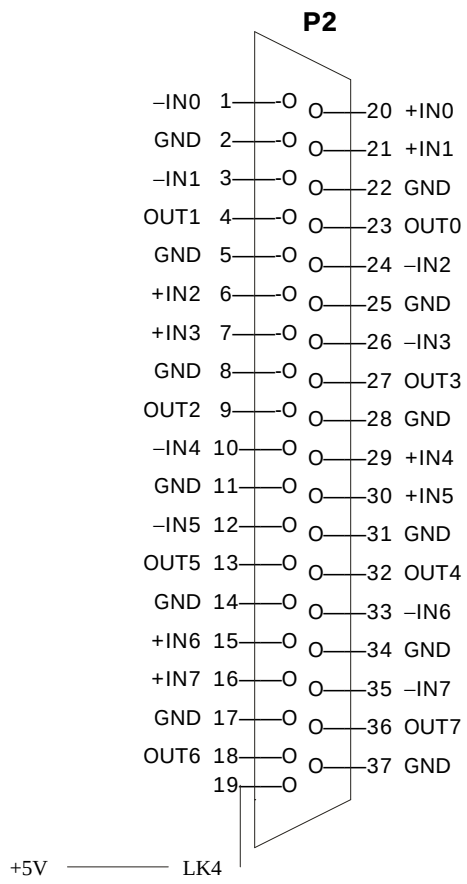


Figure 6-1. End View of DB37 Analog I/O Connector

Note that a DB37-to-phono breakout cable is available for the Sig32C-8. This cable brings each analog input and output channel out to one phono-type connector (inputs are brought out as single-ended). Breakout cables with BNC connectors are also available.

All CS4216 inputs are differential, allowing the use of screened twisted pair cables to minimize noise pick-up. Inputs have programmable gain, both internal to the CS4216 (up to 22.5 dB) and as external instrumentation amplifiers (up to additional 60 dB). Outputs have progammable attenuation internal to the CS4216 (up to -22.5 dB). Inputs are protected up to ⌠40V, and outputs are short-circuit protected.

Each CS4216 is a stereo device, containing two input and two output channels. Thus, there are four CS4216s on each Sig32C-8 board, but eight sets of everything else input/output-related: Burr-Brown instrumentation amplifiers, output followers, input resistors and overvoltage protection, etc.

All eight (8) A/D and D/A converters on the Sig32C-8 board sample simultaneously. Standard digital audio sample rates can be generated by onboard oscillators or provided externally (see section 6.1, above). When Hypersignal software is used, 16 different sampling rates are software-selectable.

6.2.1 Programmable Input Gain

In addition to some amount of programmable input gain inside the CS4216 CODECs (from 0 dB to 22.5 dB), the Sig32C-8 also contain eight (8) Burr-Brown instrumentation amplifiers that can be programmed to provide an additional 0, 20, 40, or 60 dB of input gain. Both the CS4216 gain and the external amplifier gain can be set independently for each input channel. When Hypersignal software is used, total input gain can be set as one of 16 possible values ranging from 7.5 dB to 82.5 dB for all channels independently. See table below for the possible values.

Gain Value (dB)	Gain Value (V/V)	Full-Scale Input Voltage (mV)[1]
0	1	1400
7.5	2.37	590.7
15	5.62	249.1
20	10	140
22.5	13.33	105
27.5	23.71	59
35	56.23	24.9
40	100	14
42.5	133.35	10.5
47.5	237.14	5.9
55	562.3	2.49
60	1000	1.4
62.5	1333.5	1.05
67.5	2371.4	0.59
75	5623.4	0.249
82.5	13335	0.105

Notes:

[1]For default (unity-gain) full-scale input voltage of ⁄1.4V (factory configuration). See section 6.2.2 below for information on changing the unity-gain full-scale voltage range.

NOTE: Increasing hardware gain from 0 dB will cause increased noise floor, and reduced dynamic range. Degradation will depend on the amount gain used and level and type of noise in the signal. Use high gain settings with caution!

Four external digital I/O lines provided by the CS4216 CODECs for user-defined purposes are utilized to program the external amplifiers. These lines are set as part of the output control words sent to the CS4216s between output DAC data samples. See section 6.3 below for information on setting the fields inside CS4216 output control words.

See sections 7.1 Hypersignal Software, and 7.2 DSPower Software, respectively, for examples of setting gain values in either Volts or dB in software.

NOTE: An input signal must be connected to obtain correct programmable hardware gain performance on any given Sig32C-8 input channel. Leaving input channels unconnected and attempting to use Hypersignal software instrument displays to view expected gain settings based on unconnected noise floor input will not be accurate.

6.2.2 Input Voltage Range

Default (unity-gain) full-scale voltage range for each Sig32C-8 input channel is ⁄1.4V. This can be increased by inserting a series resistor between the signal and Sig32C-8 input, which creates a voltage divider with the channel's input impedance. For example, inserting a 62 kOhm resistor in series with an input channel would result in approximately ⁄10.0V full-scale input voltage for that channel.

Using the default 10k input impedance (see section 6.2.3 below) for each Sig32C-8 input channel, the following full-scale ranges are suggested:

Series Resistor (kOhm)	Full-Scale Input (∫V)	Input Impedance (kOhm)
4.3	approx. 2.0	14.3
27	approx. 5.0	37
62	approx. 10.0	72

Notes:

1) If the default input termination has been changed from 10k, then different series resistor values are required.

2) There is one resistor pack for each two (2) channels. Thus, the input impedance and can only be set for channel pairs. The full-scale input range can be set for each channel individually, since a series resistor must be added externally.

3) To get an exact input range value, a non-standard series resistor with high precision may be necessary.

4) This technique applies to both single-ended input and differential input.

5) If you are using Hypersignal software or DSPower software, make sure to set the corresponding System Configuration menu fields (or Hardware Setup Dialog Box).

6.2.3 Input Impedance

Default input impedance for Sig32C-8 boards is 10 kOhm || 12 pF. Each CS4216 device has a socketed resistor pack (4 individual resistors each) which provided termination to ground for the V+ and V- inputs on the stereo inputs—two resistors for Left channel, and two for Right channel. Input impedance can be changed by replacing the socketed resistor packs, located in front of each pair of Burr-Brown PGA204A instrumentation amplifier ICs (eight (8) amplifers, four (4) resistor packs total). Each resistor pack is an 8-pin SIP package, located adjacent to the PGA204 amplifiers and just behind the 37-pin D-sub endplate connector.

The resistors in each of these packs provide the termination to ground for the V+ and V- analog inputs. Replacing these resistor packs with a different value (e.g. 100 kOhm) changes the input impedance to higher or lower values, as needed.

NOTE: There is only termination resistor pack for each two (2) channels. Thus, input impedance can only be changed for input channel pairs 0,1 and 2,3—not all channels independently.

The default termination of 10 kOhm to ground has been chosen as a compromise between input signal loading and noise pick-up. Note that lower impedance values decrease noise pick-up during A/D conversion, so change input impedance values with care.

6.3 CS4216 Information

The Sig32C-8 contains four (4) dual CS4216 CODECs. Each CODEC contains two A/D converters and two D/A converters. All converters use sigma-delta technology, which is based on high oversampling rates, and built-in digital filters to “downsample”, or decimate oversampled input data to the desired sampling rates, and interpolate or “upsample” output data to the final output rate. Sigma-delta technology offers the advantage of inherent anti-aliasing and reconstruction filters that track with the programmable sampling rate, sharp filter responses, and nearly perfect linear phase. Disadvantages include delay due to the digital filters, which can become excessive at low sampling rates, and ineffectiveness at very low sampling rates because the oversampled rate no longer offers analog bandlimiting protection. See the Hypersignal Hardware Reference Guide for more information about sigma-delta technology.

On the Sig32C-8 board, the CS4216 CODECs are used in SM2, or mode 2. Each CS4216 CODEC is programmable by setting bits in various “sub-frame” fields located in the lower half of the 32-bit DAC output words. Fields include muting D/A outputs, selecting ac or dc input coupling, input gain, output attenuation, and digital output pins (TTL level). Each D/A converter must be sent a 32-bit data word during each sample period. The upper 16 bits (most significant 16-bit word) contains output sample data, and the lower 16 bits CODEC control data (the programming information just described above). When serial port DMA operation is used, the DAC control words can be set beforehand, placed in the DMA area, and left untouched until a change is needed. See Appendix C, CS4216 Data Sheets, for fully detailed information on the format of control words and programming CS4216 devices.

Upon exit from reset mode (see chapter 4, CONTROL/STATUS REGISTER), the CS4216 CODECs undergo an automatic calibration process to minimize ADC and DAC offsets. See section 7.1.1.5, Source Code Initialization Examples, for source code excerpts that wait for calibration to complete before beginning operation.

7. SOFTWARE

There are five levels of software support for Sig32C System boards:

- the DSPower Windows software package provides a graphical, Windows-based user-interface, and can transparently control Hypersignal-Macro or Hypersignal-Acoustic (see next paragraph) as “DSP Engines”. DSPower offers a block-diagram interface, simulation and interactive modes, and source code generation.

- Hypersignal-Macro and Hypersignal-Acoustic are off-the-shelf DOS software packages that support the Sig32C-8 with real-time instrument functions such as Spectrum Analyzer, stimulus & response measurement, continuous signal generation, real-time disk record and playback, and more. These packages contain numerous DSP and math functions (which can in some cases be accelerated using Sig32C boards), display and visualization functions, and DSP-oriented functions such as Difference Equations, sampling rate conversion, frequency zoom, etc. Digital filters can be designed and tested in real-time, and real-time code development utilities are available to support the Hypersignal DSP32C Source Code Interface.

- the DSPower-HWLib DLL offers a Windows C/C++ program interface to Sig32C series boards. Low-level functions include board reset, run, stop, register access, DSP32C executable file (COFF file) download, memory transfer, and semaphore management. High-level functions include continuous acquisition/generation to/from Hypersignal .TIM or .WAV files, continuous function generation, stimulus & response measurement, and more. The HWLib DLL includes digital oscilloscope and digital tape recorder DLL C/C++ example programs. C/C++ programs can interface to Hypersignal DSP32C source code, or to user-defined DSP32C source, or combination of both.

- the Hypersignal DSP32C source code interface allows Hypersignal operation to be customized, by modification of, or addition to, the source code routines, DSP/math functions, analog I/O drivers, real-time sampling structures, etc. that are used in the default DSP32C programs downloaded by Hypersignal; see section 7.1.1 below, Hypersignal DSP32C Source Code Interface, for fully detailed information

- DOS C Library, debugger, and resident monitor programs. The best options are the D3EMU and D3BUG programs, from CAC (Communication, Automation, and Control). These programs provide comprehensive development support for numerous DSP32C boards, including the Sig32C series. A full range of host PC library calls, including program download, several types of memory transfer, board control, are available in the D3EMU package. Comprehensive debugging, breakpoint, register and memory printout, execution history, and other features are included in the D3BUG package. Contact either Signalogic or CAC for more information. Another option, included with Sig32C boards, is a rudimentary DOS C Library and resident monitor program. The QW3LIB C Library allows simple user-defined host PC programs to be written for Sig32C boards. Basic board control, program download, and memory transfer routines are available. The resident monitor DSP32C source code is included as a guide and starting point to writing DSP32C code to interface to user-defined host PC programs. The RESMON3 resident monitor program allows basic DSP32C operation to be monitored, by allowing host PC display of DSP32C registers and memory locations at various points in DSP32C program execution.

Note that since Sig32C boards contain the general-purpose, programmable DSP32C device, a *DSP32C program must always be downloaded to DSP32C memory* before the board can do useful work. Thus, in order to get a Sig32C board to do what you want, you must do one of the following:

- use Hypersignal and/or DSPower-HWLib DLL software, which include several DSP32C code functions that are downloaded depending on software operation, as well as built-in drivers to control and transfer programs and data to and from the board

- modify or add to existing Hypersignal DSP32C source code to change DSP32C operation, but continue to use Hypersignal and/or DSPower-HWLib DLL to download programs and control board operation (see section 7.1.1 below, Hypersignal DSP32C Source Code Interface)

- write *both* user-defined DSP32C source code *and* user-defined host PC ‘C’ programs to fully operate and communicate with the board, using the C Library

Any time you need to write DSP32C source code, you will need at least the AT&T DSP32C assembler and linker. If real-time processing constraints allow it—or the size of the project is large—the AT&T DSP32C ‘C’ Compiler is also recommended. In addition, the AT&T DSP32C Software Simulator is always recommended as a means of finding logic or design errors in algorithms. Last, but not least, the AT&T Family Application Library and Hypersignal DSP32C Source Code Interface are recommended for any level of DSP32C source code development, as they contain numerous DSP32C algorithms and functions in source and binary form, such as optimized FFTs, filters, matrix, transcendental, trig, signal manipulation functions, etc.

7.1 Hypersignal and DSPower

See Signalogic publications “Hypersignal: Defining DSP Software Today”, “DSPower Technical Description”, and “DSPower-HWLib DLL: Windows C/C++ Library”, for more information about these high level programs.

SIG32C SYSTEM BOARD USE WITH HYPERSIGNAL

If you have Hypersignal software, the following information applies to use in conjunction with Sig32C boards. This information also applies if Hypersignal is being used as a DSP Engine with DSPower software.

Basic Descriptive and Installation Details

The software instructions below are given in terms of entry into Hypersignal menus. Entry for DSPower dialog box and block setup boxes is given in parentheses.

Use a hardware designator of ‘SIG32C’ in the Hypersignal software System Configuration menu (or DSPower software DSP/Analog Hardware Setup dialog box). The 80286 INSTRUCTION SET field in the System Configuration menu (or dialog box) should be enabled—set to ‘Y’—if you have a 80286, 80386, or 80486 machine.

> NOTE: Initial default Hypersignal I/O base address for all Sig32C boards is 290H, which is the factory default. However, you may need to change this address if there are conflicts with other boards plugged into your computer, or you will be using more than one Sig32C board. See chapter 2, INSTALLATION, for more information. If you need to change the Hypersignal I/O base address, see Hypersignal Users Manual section 2.5.2, Initializing the System Configuration Menu, under the I/O BASE ADDRESS field.

External Clock

An entry of ‘-EXT’ after the entry(s) for this board in the System Configuration menu (or Hardware Setup dialog box) will disable any built-in sampling rate programming and menu-entry range checking. In this case, you would be expected to supply an external clock or non-standard crystal oscillators and ensure that a matching value is entered in the software SAMPLING FREQUENCY menu fields (or block-diagram setup boxes) in order to maintain correct time units display and usage. If you change oscillator values, then you will need to make minor modifications to the DSP32C source code in order to program the sampling rate latch correctly (because the built-in Hypersignal programming is disabled by the ‘-EXT’ selection). See section 7.1.1 below, Hypersignal DSP32C Source Code Interface.

Sampling Frequency

Sampling frequency values are set in the data-entry menus of instrument functions. Entry is in Hertz. Note that there are sixteen (16) possible sampling frequency values (see table in section 6.1, Sampling Rates). Entered values not exactly matching a possible value are quantized to the nearest possible value.

Programmable Hardware Gain

In the instrumentation functions, entry in the INPUT RANGE/GAIN fields of either full-scale voltage dB gain values is possible. Entry of voltage values can be as just numbers (in Volts), or prefixed with ‘v’. Entry of dB values must be pre-

fixed with ‘d’. Entry lists are possible in order to specify different gain settings on individual traces (or channels); separate entries with a comma. In the examples below, the unity-gain full-scale voltage range of the Sig32C-8 is assumed to be 1.4 V (factory default):

```
0.14                INPUT RANGE/GAIN:
```

The above specifies 0.14 Volt full-scale input (a gain of 10 V/V). So does the following example:

```
v0.14               INPUT RANGE/GAIN:
```

Specifying gain values in dB always requires a ‘d’ prefix. For example:

```
d10,d20             INPUT RANGE/GAIN:
```

would specify 10 dB of gain on the first trace (or channel), and 20 dB of gain on the second trace (or channel).

Note that there are sixteen (16) possible gain values (see table in section 6.2.1, Programmable Input Gain). Entered values not exactly matching a possible value are quantized to the nearest possible value.

If necessary, default (unity-gain) full-scale voltage for the Sig32C-8 can be changed by changing input impedance termination resistors and/or adding series input resistors (see sections 6.2.2, Input Voltage Range, and 6.2.3, Input Impedance).

> NOTE: An input signal must be connected to obtain correct programmable hardware gain performance on any given Sig32C-8 input channel. Leaving input channels unconnected and attempting to use Hypersignal instrument displays to view expected gain settings based on unconnected noise floor input will not be accurate.

Digital Offset and Trigger Values

Input and output offset and trigger value entries in the Analog Conversion function (or D/A Input block) are active and applied digitally, via DSP32C code. They should be entered in the range -32767 to 32767, which corresponds to 16-bit A/D converters.

Hardware Ratings and Software Throughput Specifications

Below are some specifications for a single Sig32C-8 board when used with Hypersignal software. See also section 1.3, Specifications.

The specifications are divided into 4 categories: 1) PERFORMANCE, 2) DIGITAL, 3) A/D, and 4) D/A. Categories 2-4 are basically redundant from section 1.3, Specifications, above. Category 1, PERFORMANCE SPECIFICATIONS, results from integrated testing of the board in conjunction with Hypersignal software. The machine used for this testing was an AT-class 33 MHz 386 with approximately 18 msec avg. access time hard disk and 11 MHz I/O bus. The “1 channel Hard Disk Record” test refers to the Analog Conversion function in Hypersignal software, and simply measures the maximum rate the system can stream continuous A/D data to hard disk. The “Max. N-chn Spectrum Analyzer Rate” refers to the Spectrum Analyzer function in Hypersignal software, and measures the maximum sampling rate at which the Spectrum Analyzer function will operate at before it produces invalid results, FFT overflow, or simply “locks up”.

Maximum rates are complicated values interdependent on the speed of the DSP chip, the speed of the host I/O bus, and the type of data transfer mechanism used on the board (note that the speed of the host computer video display has no impact).

The “1024 pt FFT Accel. Disk File Rate” test refers to an accelerated FFT function, measured over 500 1024 pt. frames (no zero-filling), read in from hard disk, processed by the hardware, and written back to hard disk. The “Spectrum Analyzer Noise Floor” test refers to the Spectrum Analyzer function in Hypersignal software, and is performed under the following conditions:

1) 50 kHz sampling rate (or maximum of hardware, if less)

2) single-channel displayed (log magnitude format)

3) no averaging; measurement is essentially “peak-hold”, monitoring the highest spurious level from dc to 25 kHz

4) analog input shorted to ground (or to V- input if differential input), or left unconnected—whichever produces lower spurious results

5) 1024 pt. FFT, 512 pt. framesize, Blackman window

Due to the wide variance in different machines and “clones”, even those with ostensibly identical characteristics, you may encounter machine-dependent discrepancies with the figures.

While the utmost care and effort has been made to obtain accurate measurements, Signalogic cannot guarantee their accuracy, and is not responsible for errors and omissions.

PERFORMANCE SPECIFICATIONS

DSP32C, 81.4 nsec	Digital Signal Processor:
floating-point, 32-bit	Data Format, Wordlength:
48 kHz	1 channel Hard Disk Record:
48 kHz	1 channel Hard Disk Playback:
48 kHz	Max. 2-chn. Spectrum Analyzer Rate:
19 frames/sec	1024 pt FFT Accel. Disk File Rate:
83 dB	Spectrum Analyzer Noise Floor:

DIGITAL SPECIFICATIONS

32k x 32	Standard Ext. Mem. Config:
128k x 32	Maximum Ext. Mem. Config:
no (yes for processor Synchronous)	Ext Sync. Serial Interface:
no	Ext Async. Serial Interface:
yes, QWbus	External Parallel Interface:
Prog. I/O Mapped	Host Data Transfer Method:

A/D SPECIFICATIONS

2, 16-bits	Number of linear PCM channels:
none	Codec channels:
yes, QWbus	Expandable:
yes	Simultaneous S/H
80 dB	Typical Dynamic Range:
UA	THD:
⎛1.4V to ⎛.105mV; P16	Voltage Range:
yes, SD	Anti-Aliasing Filters:
yes, P16	Programmable Sampling Rate:
yes	External Sample Clock:
48 kHz 16-bit	Max. Sampling Rate:
7.35 kHz	Min. Sampling Rate:
yes	External TTL-Level Trigger:
no	Programmable Level Trigger:

D/A SPECIFICATIONS

2, 16-bits	Number of linear PCM channels:
none	Codec channels:
yes, QWbus	Expandable:
yes	Simultaneous S/H:
80 dB	Typical Dynamic Range:
UA	THD:
⎛2.98V	Voltage Range:
yes, SD	Reconstruction Filters:
yes, SD	Sin(x)/x Filters:
yes, P16	Programmable Sampling Rate:
yes	External Sample Clock:
48 kHz 16-bit	Max. Sampling Rate:
7.35 kHz	Min. Sampling Rate:
yes	External TTL-Level Trigger:

Below is a legend for the abbreviations and acronyms used in the hardware specifications above:

AIC	analog interface circuit
BESSELn	Bessel anti-alias or reconstruction filter type, followed by value indicating order of filter
BUTTERn	Butterworth anti-alias or reconstruction filter type, followed by value indicating order of filter
CODEC	coder/decoder (normally refers to μ-Law or A-Law log companding)
CHEBYn	Cheybshev anti-alias or reconstruction filter type, followed by value indicating order of filter

DI	differential-ended analog input
DMA	direct-memory-access transfer
DS	delta-sigma D/A converter
ELLIPn	elliptic anti-alias or reconstruction filter type, followed by value indicating order of filter
Fc=nn	fixed cutoff-frequency, usually followed by a value in kHz
Jnn	jumper with nn different positions
JB	jumper-bypassable
Hz	Hertz (samples/sec)
kHz	kiloHertz (10^3 samples/sec)
MM	maximum memory configuration
msec	millisecond (10^{-6} second)
nsec	nanosecond (10^{-9} second)
OB	offset binary
PA	potentiometer-adjustable (external knob)
Pnn	programmable via software, followed by number indicating number of discrete values possible
PCM	pulse code modulation—sending values by a series of pulses with different spacing (binary transmission)
Prog. I/O	programmed I/O transfer (see SAMPLING RATE discussion previous)
RP	programmable via resistor jumper or block (not software)
SD	sigma-delta A/D converter
SI	single-ended analog input
S/H	sample-and-hold
TA	trimpot-adjustable
THD	total harmonic distortion
UA	figure unavailable
1CH	indicates maximum sampling rate must be divided by number of channels in use

In the DIGITAL SPECIFICATIONS section, “P” and “D” stand for program memory and data memory in cases where the DSP device uses a Harvard architecture (separate memory buses). In cases where the DSP device is Von Neumann (single bus), no designator is used. When no designator is used, the memory *may be both instruction and data*. In cases where there is both program memory *and* split data memory, “X” and “Y” are used to indicate the separate data memory areas.

Onboard Memory Usage

The basic Sig32C board contains 32k x 32 of zero wait-state external A area memory. Hypersignal uses less than 4k x 32 of this memory for program storage, and the remainder for analog data buffers, file-based acceleration processing, and other operational data buffer requirements. Sig32C memory can be expanded to 128k x 32 of total external A memory. The 128k x 32 amount is strongly recommended in order to fully enable all Hypersignal acceleration functions, allow continuous real-time functions to operate at full speed and capability, and increase maximum output stimulus sizes for stimulus & response measurement.

7.1.1 Hypersignal DSP32C Source Code Interface

Below please find information concerning the DSP32C Source Code Interface used by Hypersignal software. Some of the information below is given in general terms for “DSP devices” and not the DSP32C specifically, since Hypersignal supports multiple DSP chip families and various DSP hardware systems. Sections 7.1.1.2 through 7.1.1.5 speak to the DSP32C directly.

The disk containing the source code and support utilities is labeled “disk #4", and is not normally included in Hypersignal software shipments, unless purchased additionally or separately.

IMPORTANT NOTES: A signed Non-Disclosure Agreement (NDA) is required prior to shipping the DSP Source Code Interface for a given DSP chip. Bascially, the NDA requires work with the DSP Source Code to be for in-house R&D and product-development purposes, and prohibits using the DSP Source Code within, or to help develop, a product competing with Hypersignal.

In order to make effective use of this very flexible product, appropriate assemblers/linkers are required. Individual DSP types can be specified by part number; see the Hyperception and Signalogic price sheets. 'C' compilers, while not required, are suggested for low- to mid-range sampling rate applications where utmost code efficiency is not necessary.

In all cases, the DSP source code has been implemented and tested with the ***manufacturer's development tools*** (i.e. assembler, linker). Development tools from other vendors may well be compatible, but Signalogic cannot guarantee this.

The DSP Source Code Interface includes both library routines and also real-time, interrupt-driven structures that allow foreground and background processing in conjunction with analog I/O or digital I/O sample interrupts. Initially, the source code supports standard Hypersignal operation, including instrumentation displays, continuous transfer to/from disk, and file-based function acceleration, with no programming required. In other words, if you were to assemble and link the source code for a specific DSP chip as it was shipped, without any changes, you would end up with the distribution object file that is sent with every basic Hypersignal package. However, the source code could then be modified and/or added to in order to enhance the baseline Hypersignal operations or add new functions entirely. Changes can be as simple as interfacing to new analog converters or as complicated as simulating and testing complete algorithms by transferring data between the DSP board and Hypersignal software resources (including hard disk and video display) using pre-defined buffer-switching mechanisms and real-time interrupt structures.

A simple example might be to create user-written DSP source code that might perform some algorithm or feature extraction. Following this, Hypersignal can be instructed to use the new executable DSP file(s) as needed, causing the new user-written algorithm(s) to execute. This type of modification process basically allows you to concentrate on the problem at hand, since the Hypersignal high-level host software performs all of the housekeeping associated with controlling the DSP board, downloading executable code, and transferring blocks of real-data between host and DSP board memory. Furthermore, if you maintain compatibility with internal Hypersignal data and waveform file formats, then more housekeeping tasks such as file creation, graphical analysis, etc. can be left to Hypersignal.

Following are some more detailed examples of using the DSP Source Code Interface. Lets suppose that you had one of the DSP boards that is supported by Hypersignal, but needed to interface the board to a target analog section of your own design. If you had the DSP source code for the appropriate chip family, this would be trivial. First, you would change the interrupt service routine for the Hypersignal function that you wanted to get working. Depending on the chip family there is generally a few different interrupt service routines for each function, organized by board manufacturer. The changes you would make here would be minor: ensuring that the code read your A/D port address instead of the supported board's standard address. Next, you might have to change the system initialization module if the interrupt source/level for your analog hardware is different. Everything else would be left alone: buffering of the data, transfer to the high-level host software, and processing of the data (although you might want to change that, too, at some point). Next, you would assemble the modules changed using the "ASM" batch file provided for each chip family. Finally, you would invoke the "LNK" batch file, and copy the resulting linker output file (".OUT" for AT&T DSP32C development tool software) to your working Hypersignal directory. When you next invoked Hypersignal, this file would be ready for downloading and subsequent execution without any further effort.

As another example, lets suppose you wanted to take a supported board, and working with the board's standard analog I/O, implement all or part of a time domain algorithm on the board and stream the real-time results to hard disk for later evaluation and display. If you first make the assumption that when running this function from Hypersignal you will ask for two-channel operation, you could simply alter the foreground processing to apply your algorithm to every other sample in the input buffers as they become ready. This is possible because multichannel data in Hypersignal is always transferred and processed in interleaved form; indeed, the high-level host software can be instructed to acquire/generate N channel data whether the board has N channels or not. Whether this works depends totally on the low-level DSP source code. After modification of the foreground module, and/or addition of new, user-defined routines, subsequent execution

of Hypersignal with 2-channel operation specified would produce raw input data on the first channel, and processed data on the second channel. FFT operation on such a waveform would be automatic, since Hypersignal still thinks this is a “stereo” waveform, allowing convenient comparison viewing and evaluation in the frequency domain as well.

As a third example, consider using a supported board to simulate an entire project, maintaining just enough of the Hypersignal interface and real-time data transfer mechanism to allow real-time display of data, debug modes, and possible streaming of data to disk. After transferring code to your target system (via EPROM), there can be no doubt as to whether encountered problems are due to the software or the target hardware design—the Hypersignal simulation can always be easily run at any time as a check against the software design.

Other examples include adding DSP source code to support multiprocessing, interconnections between multiple boards, and other peripheral bus standards such as DT-Connect, measuring algorithms or parts of algorithms against a particular real-time sampling rate, evaluating the effects of finite word-length, and more.

NOTE: Most interrupt service routines fully save and restore machine context to protect anything they might modify, and most routines (such as the “snap-in” real-time filter routines included in the source code library) also save and restore anything they might destroy. In all chip families the Digital Oscilloscope and Spectrum Analyzer functions provide the best examples of full context-save interrupt service routines combined with substantial foreground processing constructed from many of the routines in the source code library.

7.1.1.1 Source Code License Agreement

Compliance with the following license agreement is one of the provisions in the NDA agreement mentioned above, a signed copy of which is required before delivery of a DSP Source Code Interface can take place:

Hyperception, Inc. and Signalogic, Inc. reserve all rights to source code supplied with “Hypersignal” series software.

USE OF THIS SOURCE CODE BY YOU (“YOU” MEANS THE COMPANY AND/OR INDIVIDUALS THAT ARE USING THE “HYPERSIGNAL” SOFTWARE) FOR PURPOSES OF RESALE OR PROFIT, OR FOR INCLUSION IN, EITHER WHOLE OR IN PART, ANY OTHER SOFTWARE INTENDED FOR SALE OR PROFIT, IS PROHIBITED.

You are free to copy source code sections to make backup copies and to perform experimentation and testing during algorithm and product development and research. If it is discovered that you, through resale, through lending, or allowing through carelessness or inaction, have enabled another party to obtain possession of a section of the DSP source code marked with a Hyperception or Signalogic copyright, which the other party is including within, or with, a product competitive with the “Hypersignal” series software, then HYPERCEPTION AND SIGNALOGIC ARE PREPARED TO PROSECUTE TO THE FULLEST EXTENT OF THE LAW.

YOU ACKNOWLEDGE THAT YOU HAVE READ THE ABOVE AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT AND THE NON-DISCLOSURE AGREEMENT (see above) IS THE COMPLETE AND EXCLUSIVE STATEMENT OF ANY AGREEMENT BETWEEN US, INCLUDING ANY OTHER ORAL OR WRITTEN COMMUNICATION.

7.1.1.2 Source Code Overview

Disk #4, if specified in the software purchase order, contains all AT&T DSP32C source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic DSP32C source code includes interrupt service routines and analog I/O drivers for the Ariel, AT&T, CAC, MTT Instrumentation, LSI/Spectrum, National Instruments, and Signalogic boards.

As is, the DSP32C source code is written in modular form, and supports the basic “modes” of operation below:

Module	Hypersignal Function	oper. “mode”

MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1
MODUL2	Digital Oscilloscope	mode=2
MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7
MODULn	User-Defined function	mode=n

INTERRUPT SERVICE ROUTINES (ISRs)

There are various interrupt service routines that respond to interrupts on the DSP32C boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of “ISRn”, where n indicates the operating mode (see above) that the routine supports, for example “ISR0” for operating mode zero. Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization (“itable” and PCW register) in RESET.S and W32BEG.S to locate the correct interrupt service routine for a particular function.

7.1.1.3 Source Code and Support File List

Source code and related files contained in the Hypersignal DSP32C Source Code Interface include:

RESET.S	reset vector, wait-state, and “page register” initialization
W32DEF.S	variable & constant definition module
W32BEG.S	initialization, including ISR vectors
ISR0.S	A/D to disk ISR for LSI/Spectrum system, Ariel DSP-32C and PC-32C, CAC DSP32C-AT and AC5, National Instruments AT-DSP2200, AT&T DEV-xx, Burr-Brown ZPB34, Signalogic Sig32C-8, and MTT Instrumentation Lory DSP4200 boards
ISR1.S	D/A from disk ISR for LSI/Spectrum system, Ariel DSP-32C and PC-32C, CAC DSP32C-AT and AC5, National Instruments AT-DSP2200, AT&T DEV-xx, Burr-Brown ZPB34, Signalogic Sig32C-8, and MTT Instrumentation Lory DSP4200 boards
ISR2.S	Digital Oscillscope ISR for LSI/Spectrum system, Ariel DSP-32C and PC-32C, CAC DSP32C-AT and AC5, National Instruments AT-DSP2200, AT&T DEV-xx, Burr-Brown ZPB34, Signalogic Sig32C-8, and MTT Instrumentation Lory DSP4200 boards
ISR5.S	Spectrum Analyzer ISR for LSI/Spectrum system, Ariel DSP-32C and PC-32C, CAC DSP32C-AT and AC5, National Instruments AT-DSP2200, AT&T DEV-xx, Burr-Brown ZPB34, Signalogic Sig32C-8, and MTT Instrumentation Lory DSP4200 boards
ISR6.S	Stimulus & response ISR for LSI/Spectrum system, Ariel DSP-32C and PC-32C, CAC DSP32C-AT and AC5, National Instruments AT-DSP200, AT&T DEV-xx, Burr-Brown ZPB34, Signalogic Sig32C-8, and MTT Instrumentation Lory DSP4200 boards
OUT_TX.S	analog I/O output ISR for boards that require both Rx and Tx serial port interrupts (for boards that are unable to use only Rx; typically this is caused by Rx and Tx frames not being synchronized at low sampling rates)
MODUL0.S	foreground A/D to disk processing (mode=0)
MODUL1.S	foreground D/A from disk processing (mode=1)
MODUL2.S	foreground Digital Oscilloscope processing (mode=2)
MODUL3.S	FFT file-based acceleration processing (mode=3)

MODUL4.S	iFFT file-based acceleration processing (mode=4)
MODUL5.S	foreground Spectrum Analyzer process (mode=5)
MODUL6.S	foreground stim & response process (mode=6)
MODUL7.S	convolution/correlation file-based acceleration (mode=7)
ARCTAN.S	arctangent routine
CONVLV.S	convolution/correlation routine
DIV.S	divide routine
FFT.S	variable N (N=2^FFT_order) FFT routine
FFTOUT.S	post-FFT processing (mag, phase, etc.)
FILTER.S	filter control routine for real-time filtering
FIR.S	FIR filter routine
IIR2.S	IIR (cascade sections, modified direct form) filter routine
IIR3.S	IIR (cascade sections, balanced pole-zero structure) filter routine
LOG10.S	log (base 10) approximation routine
MAXFIND.S	find maximum-in-buffer routine
MINFIND.S	find minimum-in-buffer routine
MAG.S	magnitude routine (complex input)
PHASE.S	4-quadrant phase routine ($-\pi$ to $+\pi$; complex input)
SPAOUT.S	post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
SQRT.S	squareroot routine
WINDOW.S	apply pre-defined window (used with FFTs)
XFER.S	complex-ratio routine used for transfer function
ASM.BAT	batch file used to assemble any source module: type “ASM filename” (no extension needed)
LNK.BAT	batch file used to link all source modules: type “LNK”
W32LINK.CTL	link control file (command file for linker)

Also, disk #4 initially contains all .O files necessary to produce complete baseline, default executable code module when the W32LINK.CTL control file (above) is invoked with the AT&T d3ld utility.

7.1.1.4 Source Code Modification Process

> NOTE: Be sure to save a backup copy of the DSP32C.OUT with a different name or in a different location, in case you happen to copy over this file in the process of creating new DSP executable code files.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to DSP32C source code, keeping in mind the necessary shared variable links between Hypersignal and the DSP32C (see discussion above and W32DEF.S module, which is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .O (object) file is generated for each altered module. For convenience, an “ASM.BAT” batch file has been provided that you may use in the form of:

ASM filename <car ret>

to assemble source code modules (*do not enter a “.S” extension after “filename”*). This file executes a

d3as -Q -l %1.s

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the W32LINK.CTL control file (use an ASCII text-editor to update the W32LINK.CTL file).

3) Re-link all DSP32C modules using the AT&T DL3D linker. A “LNK.BAT” batch file has been provided that you may use by invoking:

LNK <car ret>

to link all necessary .O files. LNK.BAT executes the following statements:

```
D3LD -a -m -o DSP32C.OUT W32LINK.CTL > W32LINK.MAP
@COPY W32LINK.MAP+EOF W32LINK.MAP
@D3NM -f DSP32C.OUT > TEMP
@COPY W32LINK.MAP+TEMP W32LINK.MAP
@DEL TEMP
```

where “W32LINK.CTL” is an ASCII linker-control file that specifies all .O files to be linked by the d3ld.exe program, and W32LINK.MAP is a map file showing variable and module addresses and lengths (W32LINK.MAP is created using DOS file-redirection commands, since the AT&T linker does not support a map file output option directly; see notes below). Note that W32LINK.CTL places the reset vector module at zero, and shared variables at 0x100, forcing all relocatable code to start after the variable declarations. All code has been tested using version 1.3.1 of d3as and d3ld.

4) Copy the “DSP32C.OUT” file to the directory with the Hypersignal HSxxxxx.OVR and HSxxxxx.EXE executable files (or change the name first, copy the file, and set the System Configuration menu to use the new filename in the desired functions).

5) Run Hypersignal, and invoke the appropriate functions. Hypersignal reads the “.OUT” file (which is in “COFF” format) directly, downloads all code and data, and begins execution at the reset vector (location zero).

MISCELLANEOUS NOTES ABOUT CHANGING DSP32C SOURCE CODE

1) Hypersignal DSP32C code uses r19 to store a “page register value” (initially written by the high level software to DSP32C onchip RAM address 00FCH) so that Hypersignal DSP32C code can execute on different DSP32C boards that use different memory modes (which sometimes causes instructions to be in different 64K segments). *You should not change r19 at any time, for any reason.*

2) Hypersignal DSP32C code uses r18 as a system stack pointer for both foreground and interrupt processing code. When storing CAU registers on the “stack”, use the following sequence:

```
*r18++ = rN1e
*r18++ = rN2e
    -
    -
    -
*r18++ = rNNe
```

When restoring CAU registers from the stack, use the following sequence:

```
rN1e = r18 - 4
rNNe = *rN1—
```

7.1.1.4 Source Code Modification Process

-
-
-

```
rN2e = *rN1—
rN1e = *rN1
r18e = r18 - K        /* K = 4 x (number of registers stored) */
```

There are three remarkable things about this process. First, the stack is always expected to increment and decrement in 4-byte chunks (i.e save/restore of accumulators or extended CAU registers). Second, the stack is always expected to point to the *next available (free) location*. Third, when restoring, you can't simply do an r18e = r18 - 4 instruction and begin popping rXX = *r18— instructions, because an interrupt might occur sometime after the r18 - 4 instruction and before you finish popping; the interrupt routine, if it also used r18 as a stack pointer (which all types of routines should be able to do with a true stack pointer), might then clobber a stored register (or more than one) that hadn't been popped yet.

3) Since AT&T dedicates r20-r22 to special purposes, and Hypersignal DSP32C code uses r19 and r18 as system memory-page register and stack pointer, respectively, this means that *r18-r22 are reserved in the Hypersignal DSP32C source code and should not be used arbitrarily for any reason, ever.*

4) Hypersignal DSP32C source code is built with the "ASM" and "LNK" batch files, which invoke the AT&T "d3as" and "d3ld" assembler and linker utilities, respectively. By examining the ASM and LNK batch files, you should be able to construct an equivalent method of building the code using the AT&T "make" utility instead; however, producing a memory map file will be difficult. Obscure DOS commands have been used to create a map file in the LNK.BAT file, since the AT&T utilities will only print map output to the screen. The map file produced has a filename of "W32LINK.MAP"; you can use this file to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause.

5) Because the different DSP32C boards supported use different memory modes, Hypersignal DSP32C source code does not specify any memory mode of operation. In fact, the "LNK" batch file invocation of the AT&T linker (d3ld) does not specify any memory mode or configuration, causing the linker to assume contiguous memory starting at zero. This is possible because:

- The Hypersignal DSP32C source code always assumes at least 800H of read/write memory available at location zero (true for all memory modes). This area is used for the reset vector, storage of the "page register" value, storage of shared variables between host PC and DSP32C, storage of most DSP32C local variables, and system stack.
- Use of pc-relative "pccall" and "pcgoto" instructions that interpret procedure entry points and branch destinations as *offsets from the current pc (program counter) value*, rather than absolute constants.
- Use of r19 as a pre-defined "page register" for setting the upper 8 bits of registers being used to access local variables and interrupt table vectors not inside the 0-7FFH memory area. See discussion in 1), above.

6) The DSP32C apparently has a bug in the "call(rNN)" instruction in that only the least-significant 16 bits of the current program counter value are stored in rNN before the branch is taken. This is why all Hypersignal DSP32C code "pccall(rNN)" instructions perform a rNNe = rNN | r19 (logical "OR" with r19) as their latent instruction. It is strongly recommended that any calls you add to the source code do likewise.

7) In addition to setting them directly in the DSP32C source code, IOC register settings can be controlled from the high-level software in order to accommodate non-standard analog I/O connected to DSP32C serial ports. Inside the Hypersignal configuration file (HSMACRO.CNF or HSACOUS.CNF) there is a 32-bit integer IOC value for every DSP32C board currently supported by Hypersignal—see the "CONFIG.DOC" file on disk #2 for a list of the sequence of variables in the configuration file. Depending on the board currently in use, one of these values is sent down to the DSP32C board whenever a Hypersignal-Macro or Hypersignal-Acoustic real-time function requiring analog I/O is invoked. The corresponding DSP source code variable is "IOC_reg", which is normally used directly to set the IOC register (unless changed by the user in the DSP source code).

8) If you need to use an external sampling clock, follow the basic guidelines below:

a. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.

b. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, it is suggested that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

c. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

d. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

ACCESS AND CONTROL OF USER-DEFINED DSP32C SOURCE CODE FROM HYPERSIGNAL

The Hypersignal-Macro family of software packages and the Hypersignal macro language contain features and functions that facilitate access and control of user-defined DSP source code, such as:

1) The ability to specify different executable DSP files as the file to load and run when a DSP-based function or display is invoked from Hypersignal. In the System Configuration menu, entering a filename, such as "new.out", after the board designator will override the Hypersignal default filename.

2) A macro language function (WriteDSPVal) that allows a macro to cause Hypersignal menu and internal values to be written to arbitrary locations in DSP memory.

3) A macro language function (WriteDSPMem) that allows a macro to cause arbitrary values to be written to arbitrary locations in DSP memory.

For more information about the latter two functions, see Hypersignal Users Manual section 7.10, Macro Commands and Functions.

Note that both the WriteDSPMem and WriteDSPVal functions always make full-word accesses, depending on the natural word-length of the DSP device. This can affect the address value used, if the DSP is capable of accessing data pieces smaller than full-word length. Also, note that both functions are capable of accessing all memory types supported by the DSP chip. For a complete list of internal Hypersignal values that can be written to arbitrary DSP source code memory, see Appendix A in the Hypersignal Hardware Reference Guide, WriteDSPVal Constants, and for fully detailed information about the syntax and usage of these functions, see Hypersignal Users Manual section 7.10, Macro Commands and Functions.

For DSP32C boards, you should avoid using the program memory area between 100H and 2FFH, which is used by Hypersignal to store host-DSP interface variables and structures (100H-1FFH) and local DSP variables (200H-3FFH). Also, note that location 0FCH is used to store the page register value referred to above. You can use this value to cause program code to execute in any memory area that physically exists on the DSP32C board.

7.1.1.5 Source Code Initialization Examples

Below is an excerpt from the Hypersignal DSP32C Source Code Interface intended to show how to initialize Sig32C System boards:

```
.rsect    "reset"
/*************************************************************************/ /*      Power-on reset—start of executable code at location zero /*************************************************************************/
start:          2*nop                   /* Don't move these NOPs! */
                r5 = pcw
                r4e = 0x01              /* Signalogic Sig32C-8: disable int's, 0 wait-states for ext A, 1 wait-state for ext B */
                nop
                pcw = r4
                nop                     /* put value in pcw */
```

7.1.1.5 Source Code Initialization Examples

Below is an excerpt from the Hypersignal DSP32C Source Code Interface intended to show how to initialize the Sig32C-8 analog interface and prepare for real-time operation:

```
/*********************************/
/* Signalogic Sig32C-8 setup
/*********************************/
sig_setup:      r1e = itable            /* Sig32C-8 analog I/O uses INTREQ1 external interrupt, which indicates a frame of data (8
                                        channels) has been processed */
                r2 = r2 - r1            /* calculate offset from int. table vector to interrupt entry point */
                r2 = r2 - 8             /* account for pc+8 in relative pcgoto */
                r1e = r1 | r19          /* OR in page value (upper 8 bits of address) */
                *r1 = r2                /* store isr entry point offset in lower 16 bits of interrupt table vector */
                r4e = outdma            /* clear output data area */
                r7 = 0
                r5e = 6

clrlp:          *r4++ = r7
                *r4++ = r7
                if (r5—=0) pcgoto clrlp
                nop

                r6e = in_clist
                r7e = out_clist
                r2e = *channel          /* get channel list */
                r8e = *gainlist         /* get input gain list */
                r5e = 6                 /* extract 8 channels from list, store corresponding DMA buffer addresses to read/write */

chnlp:          r3e = r2*2              /* extract next channel in list */
                r3e = r3*2              /* make into byte offset for DMA table */
                r3e = r3 & 0x1F
                r4e = (indma+2) + r3    /* add DMA input buffer offset */
                *r6++ = r4e             /* store for use by ISRs */
                r3e = r3 + 24           /* add shift for output DMA buffer */
                r3e = r3 & 0x1F
                r4e = (outdma+2) + r3   /* add DMA output buffer offset */
                *r7++ = r4e             /* store for use by ISRs */
                r4e = r8
                r4e = r4 & 0x0f
                r13e = gaintab + r4     /* get address of gain lookup table, add gain list value as offset into table */
                r9e = 0
                r10e = 0
                r4e = r2
                r4e = r4 & 0xf          /* look at current channel; */
                r4 & 1                  /* channel even (left), or odd (right) ? */
                if (ne) pcgoto rword
                r9l = *r13              /* get actual gain val from lookup table */

lword:          r4e = outdma + r3       /* left word internal input control here */
                r1e = (outdma+4) + r3   /* left word external control here */
                r10 = r9
                r9 = r9 & 0xf0          /* mask to set left internal gain */
                r10 = r10*2             /* shift to set left external gain */
                r10 = r10*2
                r10 = r10 & 0x0c
                pcgoto setgain
                nop

rword:          r4e = (outdma-4) + r3   /* right word internal input control here */
                r1e = outdma + r3       /* right word external control here */
                r10 = r9
                r9 = r9>1               /* shift to set right internal gain */
                r9 = r9>1
                r9 = r9>1
                r9 = r9>1
                r10 = r10 & 0x03        /* mask to set right external gain */

setgain:        r11 = *r4
                r12 = *r1
```

```
                    r11 = r11 | r9          /* set internal gain (from 0 to 22.5 dB) */
                    r12 = r12 | r10         /* set external gain (20 to 80 dB) */
                    *r4 = r11
                    *r1 = r12
                    r2e = r2>1              /* shift channel list right 3 (next entry) */
                    r2e = r2>1
                    r2e = r2>1
                    r8e = r8>1              /* shift gain list right 4 (next entry) */
                    r8e = r8>1
                    r8e = r8>1
                    r8e = r8>1
                    if (r5—=0) pcgoto chnlp
                    nop

                    ioc = 0x30CC0           /* temporary ioc value; no DMA */
                    r1 = 0
                    r2e = 0xE00003          /* address of Control/Status register (byte) */
                    *r2 = r1l               /* reset codecs completely */
                    8*nop
                    r2e = chanAB            /* 32-bit scratch location */
                    r5e = 0
                    *r2 = r5e
                    nop
                    obufe = *r2             /* init output buffer to zero */
                    nop
                    obufe = *r2             /* init output buffer to zero */
                    nop
                    *r2 = ibufe             /* read one input word */
                    nop
                    *r2 = ibufe             /* read again, for the hell of it */

                    r1 = *Fs_mode           /* get host-calculated sampling clock/mode set:  5-4=ext/int clock, 3-0 samp rate setting */
                    r2e = 0xE00003          /* address of Control/Status register (byte) */
                    *r2 = r1l               /* activate codecs, set sampling rate */

                    r2e = chanAB
                    r5 = 0
                    r4 = 0
waitibf:            if (ibe) pcgoto waitibf /* wait until input buffer full */
                    nop
                    *r2 = ibufe             /* read input word */
                    r4 - 0
                    if (ne) pcgoto waitibf  /* see if skip flag is set */
                    r4 = 0
                    r3 = *r2                /* look at lower half of ADC word */
                    nop
                    r3 & 0x400              /* isolate ADV data bit */
                    if (ne) pcgoto dvalid   /* was data valid? */
                    nop
                    r5 = 0                  /* no, reset consecutive sample counter */
                    pcgoto waitibf
                    nop
dvalid:             r4 = 1                  /* set “skip” flag (next sample, right word, does not contain valid data bit) */
                    r5 = r5 + 1
                    r5 - 8
                    if (ne) pcgoto waitibf  /* loop for 8 consecutive samples */
                    nop
wait2:              if (ibe) pcgoto wait2   /* wait until input buffer full */
                    nop
                    *r2 = ibufe             /* read input word */
                    nop
                    r3 = *r2                /* look at lower half of ADC word */
                    nop
                    r3 = r3 & 0xC           /* isolate channel indicator */
                    r3 - 0xC
```

```
if (ne) pcgoto wait2        /* wait for channel 7 */
nop                         /* final initialization:  prepare for serial I/O DMA, INTREQ1 interrupts */
r20e = indma                /* set serial I/O DMA pointers */
r21e = outdma

pccall IOC_set(r15)         /* set IOC register with IOC_reg value sent from host (0x3CCC0 default, includes DMA) */
nop
r3 = pcw
nop
r3 = r3 | 0x8000            /* enable INTREQ1 (frame sync) interrupt for Sig32C-8 board */
pcw = r3
nop
return(r14)
nop
```

```
/* Gain value lookup table for Sig32C-8.  Each byte is a table entry.  Index into table comes from high-level software, in the form of one of 16
possible gain values, ranging from 0 to 82.5 dB.  A combination of the internal CS4216 gain and the external prog. amplifier gain is used to
achieve most of the values.  20 dB, 40 dB, and 60 dB are achieved using only the external gain amps.  High nibble of byte is internal gain value
(for CS4216 codec), and low nibble is external gain value (prog. differential amplifiers).  Note that both of these nibbles have to be either shifted
or masked before being stored in the DAC output area depending on whether the setting is intended for left or right channels, because CS4216 is
a stereo part and one control word must serve both channels.  See the Sig32C-8 initialization code above. */

gaintab:    int   0x5000
            int   0x01c0
            int   0x51f0
            int   0x02c1
            int   0x52f1
            int   0x03c2
            int   0x53f2
            int   0xf3c3
```

7.2 Debuggers and Monitor Programs

As mentioned at the beginning of this chapter, the best suggestion for a comprehensive Sig32C debugger are the CAC D3BUG and D3EMU packages. Both packages are symbolic debuggers, allowing DSP32C code to execute at full speed or in single-step mode. They display assembly code, registers and accumulators, memory, and breakpoints. Displays are automatically updated during single-stepping or whenever a breakpoint is encountered.

D3BUG has the additional advantage of being able to display C source statements generated by the AT&T DSP32C C compiler as the DSP32C code is being executed. Global and local variables are displayed alphabetically; assembly language statements for each C line can be displayed also.

Both D3EMU and D3BUG include DOS C callable libraries to communicate and control the Sig32C board. Several programs such as FFT, matrix, and audio demos are included as a starting point for user-defined applications.

Sig32C System boards come with the rudimentary RESMON3 DSP-resident monitor program, and some example programs that coexist with RESMON3. The resident monitor enables allows the QW3 host PC monitor program to be run, which is capable of examining DSP32C registers and memory locations, and controlling user-written DSP32C programs. QW3 can download programs, make memory transfers, set breakpoints, and read/write individual board registers.

7.3 DOS C and Windows C/C++ Libraries

As mentioned at the beginning of this chapter, the best suggestion for a comprehensive DOS C Library for interfacing user-defined C programs to Sig32C boards are the CAC D3EMU and D3BUG packages. Either package is shipped with a C callable library that includes board initialization and control, memory transfer, program download, and register access. C code examples have been tested with both Borland and Microsoft compilers.

For Windows C/C++ development, the DSPower-HWLib DLL is recommended. Below is an excerpt from the hwlib.h include file which gives programs access to the DLL. Note the inclusion of high-level functions that allow C/C++ programs to avoid interfacing to board registers or handling individual data buffers.

```
// hwlib.h:  definitions for use with DSPower-HWLib DLL


// supporting type definitions

typedef HGLOBAL HENGINE;              // engine handle
typedef HGLOBAL HBOARD;               // board handle
typedef HGLOBAL HPROC;                // processor handle

typedef char PATHSTR[144];            // path string (subdirectory + filename)
typedef char FIELDSTR[33];
typedef char FILESTR[13];

#define MAXCHAN 8                     // current maximum number of channels

typedef struct CONVERSIONINFO {       // structure used in high-level function calls (DSAcquireWvfrFile, DSPlaybackWAVFile, etc.)

// NOTES:

// 1) entire structure should be initialized to zero first, so that default values can take
//      effect for parameters not used

// 2) Current value in DSSetWaveformPath is used for waveform filename, unless
//     explicit path information is found in the filename

// 3) TrigDelay parameter not yet operational.


// basic information

    PATHSTR   szFilename;             // [path]filename of file being acquired/generated
    short int NumChan;
    short int ChanList[MAXCHAN];      // channel list; can contain ‘,’ and ‘-’ other characters to specify channel lists and ranges
    float     Fs;                     // sampling rate (in Hz)
    DWORD     NumSamples;             // number of samples to acquire/generate
    FIELDSTR  szGainList;             // string specifying gain list for input channels
    long      Offset;                 // offset added to data (in A/D counts, can be +/-)
    BOOL      DALooping;              // enables/disables continuous D/A looping; 0 or 1, valid only during generation (output)
    BOOL      Monitor;                // enables/disables D/A loopback when recording; 0 or 1, valid only during acquisition (input)
    BOOL      FileSplit;              // 1 enables A/D file-splitting for multichannel, 0 disables
    DWORD     Frmsiz;                 // framesize (stored in waveform header; used for post-processing only)

// real-time digital filter information

    FILESTR   szFilter1;              // name of real-time digital filter file; can contain ‘/’ character
    FILESTR   szFilter2;              // to allow dual-filter entry (filter files must be located on same path as waveform)

// trigger information

    WORD      TrigMode;               // trigger mode (0 = internal, 1 = external)
    long      TrigLevel;              // trigger level (in A/D counts, can be +/-)
    long      TrigDelay;              // delay from trigger relative to data (in sec, can be +/-)
    short int NumTrigChan;
    short int TrigChanList[MAXCHAN];// trigger channel list

// stimulus & response information

FILESTR   szStimName;             // name of stimulus output file; used only for DSStimRespMeasure function (must be located on
                                  same path as waveform)
WORD      StimMode;               // controls stimulus usage in Stimulus/Response function
long      StimDelay;              // delay from stimulus onset to start of record (+/- value)
};

typedef CONVERSIONINFO FAR* PCONVERSIONINFO;


// DLL call prototypes

extern “C” {

// engine functions
```

```
HENGINE     WINAPI DSEngineOpen (UINT, LPCSTR, LPCSTR);
UINT        WINAPI DSEngineClose (HENGINE);
UINT        WINAPI DSGetEngineErrorStatus (HENGINE);
UINT        WINAPI DSGetEngineUsage ();
UINT        WINAPI DSSendEngineCommand (HENGINE, UINT);
UINT        WINAPI DSRegisterEngineMsgWnd (HENGINE, UINT, HWND);
UINT        WINAPI DSUnregisterEngineMsgWnd (HENGINE, UINT);
UINT        WINAPI DSSetWaveformPath (HENGINE, PATHSTR);
UINT        WINAPI DSSetTemplateFilePath (HENGINE, PATHSTR);

// board initialization and control functions (all processors simultaneous)

HBOARD      WINAPI DSAssignBoard (HENGINE, LPCSTR, UINT, UINT, UINT);
UINT        WINAPI DSFreeBoard (HBOARD);

UINT        WINAPI DSInitBoard (HBOARD);
UINT        WINAPI DSResetBoard (HBOARD);
UINT        WINAPI DSRunBoard (HBOARD);
UINT        WINAPI DSDisableBoard (HBOARD);
UINT        WINAPI DSHoldBoard (HBOARD);


// individual processor initialization and control functions

HPROC       WINAPI DSAssignProcessor (HBOARD, short int);
UINT        WINAPI DSFreeProcessor (HPROC);

UINT        WINAPI DSLoadFileProcessor (HPROC, LPCSTR, short int);

UINT        WINAPI DSResetProcessor (HPROC, short int);
UINT        WINAPI DSRunProcessor (HPROC, short int);
UINT        WINAPI DSDisableProcessor (HPROC, short int);
UINT        WINAPI DSHoldProcessor (HPROC, short int);


// memory and variable transfers

UINT        WINAPI DSPutBoardMem (HBOARD, UINT, DWORD, UINT, void far*, DWORD);
UINT        WINAPI DSGetBoardMem (HBOARD, UINT, DWORD, UINT, void far*, DWORD);
UINT        WINAPI DSPutHVarMem (HBOARD, DWORD, DWORD);
DWORD       WINAPI DSGetHVarMem (HBOARD, DWORD);


// wait-for-buffer and wait-for-flag

UINT        WINAPI DSWaitForHBuffer (HBOARD, short int, HWND);
UINT        WINAPI DSWaitForHFlag (HBOARD, short int, DWORD, HWND);
UINT        WINAPI DSCancelWait (HBOARD);


// interrogate board system parameters

UINT        WINAPI DSGetBoardCode (HBOARD);
double      WINAPI DSCalcSampFreq (HBOARD, short int, float, float);
DWORD       WINAPI DSGetMemSize (HBOARD, short int);
UINT        WINAPI DSGetBoardMemArch (HBOARD);


// get/set board base address, instruction set, processor num, registers, etc.

short int   WINAPI DSGetBoardIOBaseAddr (HBOARD);
short int   WINAPI DSGetBoardMemBaseAddr (HBOARD);
UINT        WINAPI DSSetBoardIOBaseAddr (HBOARD, short int);
UINT        WINAPI DSSetBoardMemBaseAddr (HBOARD, short int);
short int   WINAPI DSGetProcessorNum (HBOARD);
UINT        WINAPI DSSetProcessorNum (HBOARD, UINT);

UINT        WINAPI DSGetInstructionSet (HBOARD);
UINT        WINAPI DSSetInstructionSet (HBOARD, UINT);

UINT        WINAPI DSReadBoardReg (HBOARD, short int);
UINT        WINAPI DSWriteBoardReg (HBOARD, short int, short int);

UINT        WINAPI DSRestoreDefaults (HBOARD);

// high-level functions
```

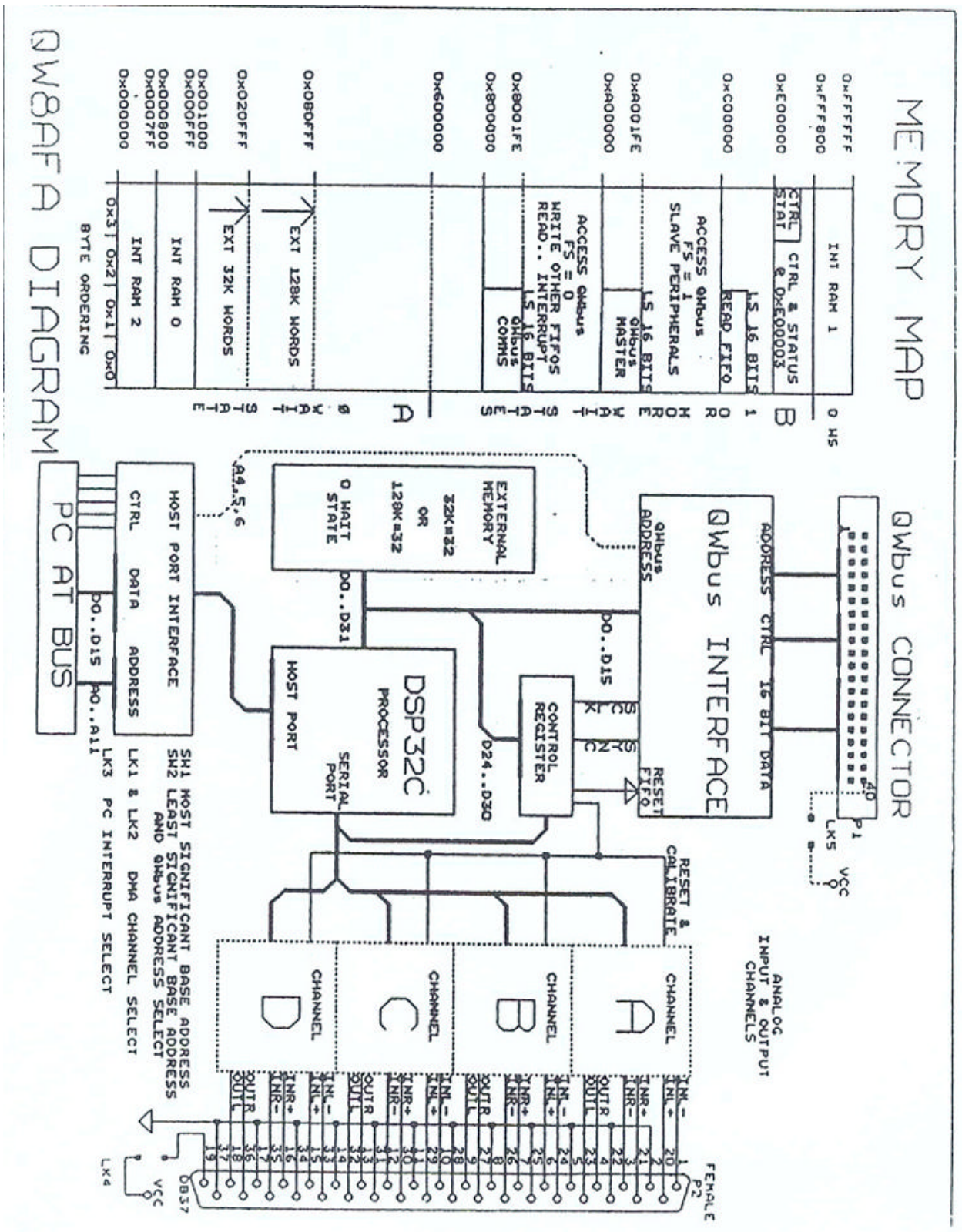
```
UINT        WINAPI DSAcquireWvfrmFile (HBOARD, PCONVERSIONINFO);
UINT        WINAPI DSGenerateWvfrmFile (HBOARD, PCONVERSIONINFO);


// engine and DLL error status

void        WINAPI DSDisableHWLibStatusWindow ();
void        WINAPI DSEnableHWLibStatusWindow ();
UINT        WINAPI DSGetHWLibErrorStatus ();

void        WINAPI DSHWLibDisplayMessage (LPCSTR);

}

// user defined messages

#define WM_DSPENGINE_BUFRDY                 WM_USER
#define WM_DSPENGINE_FLGRDY                 WM_USER+1


// error codes

#include “dllerr.h”                 // general DSPower DLL error messages

#define DSDSPFILENOTFOUND                   2
#define DSHARDWARENOTRESPONDING             1
#define HENGINE_ERROR                       -1
#define DSINVALIDENGINEPATH                 -2
#define DSENGINETIMEOUT                     -3
#define DSCOULDNOTOPENPIPE                  -4
#define DSCOULDNOTSTARTENGINE               -5
#define DSNOENGINESOPEN                     -6
#define DSNOENGINESFOUNDINLISTI             -7
#define DSENGINEHANDLEWASNULL               -8
#define DSENGINEMEMORYDISCARDED             -9
#define DSCOULDNOTINITIALIZEHWLIST          -10


// memory types returned by GetBoardMemArch

#define DS_GBMA_LINEAR          1
#define DS_GBMA_VECTOR          2


// memory types for DSGetBoardMemory

#define DS_GBM_VECTOR_DATA_X            1       // vector “X” data memory
#define DS_GBM_VECTOR_DATA_Y            2       // vector “Y” data memory
#define DS_GBM_LINEAR_DATA_RT           3       // linear “D1" real-time data memory
#define DS_GBM_LINEAR_DATA_STATIC       4       // linear “D0" static data memory
#define DS_GBM_LINEAR_PROGRAM           5       // linear “P” program memory


// bit-width constants for DSGetBoardMem & DSPutBoardMem

#define DS_GBM_SIZE8        1       // 8-bits
#define DS_GBM_SIZE16       2       // 16-bits
#define DS_GBM_SIZE24       3       // 24-bits
#define DS_GBM_SIZE32       4       // 32-bits


// possible default engine types for use in DSEngineOpen call

#define DS_EO_HSM           1       // Hypersignal-Macro 4.x (real-time)
#define DS_EO_HSA           2       // Hypersignal-Acoustic 4.x
#define DS_EO_HSMNRT        3       // Hypersignal-Macro NRT 4.x (non-real-time)
#define DS_EO_MAT           4       // MATLAB 4.x
#define DS_EO_USER          99      // user-defined


// AssignBoard constants

#define DS_AB_ENABLE8086            1
#define DS_AB_ENABLE80x86           2


// SendEngineCommand constants

#define DS_SEC_ABORT            1
#define DS_SEC_IDLE             2
```

```
#define DS_SEC_PAUSE                3
#define DS_SEC_RESUME               4


// RegisterEngineMsgWnd constants

#define DS_REMW_SETENGINEMSG                0x0001    // engine status messages
#define DS_REMW_SETSTATSMSG                 0x0002    // DSP/analog hardware statistics/parameters messages
#define DS_REMW_SETBUFNUMMSG                0x0004    // current buffer count messages
#define DS_REMW_SETLEVELMSG                 0x0008    // current data level messages
#define DS_REMW_SETALLMSG                   0x00ff    // set all messages to one control


// types of messages which can be received by registered user windows
#define DS_REMW_ENGINEIDLEMSG               0x9001    // engine status messages
#define DS_REMW_ENGINEFILEERRMSG            0x9002
#define DS_REMW_ENGINECOMPILEERRMGS         0x9003
#define DS_REMW_BUFSIZEMSG                  0x9010    // statistics/parameters messages
#define DS_REMW_SAMPFREQMSG                 0x9011
#define DS_REMW_BUFNUMMSG                   0x9020    // current buffer number message
#define DS_REMW_LEVELMSG                    0x9030    // current level message


// identifiers that can be used in PutHVar and GetHVar calls; see Hypersignal
// Hardware Reference Guide Appendix A for more information

#define DSPBOARDTYPE                0x42F       // Current Baord Selection
#define DSPOPMODE                   0x430       // Operating Mode
#define DSPFLAG                     0x431       // DSP Busy Flag
#define DSPOVERFLOW                 0x432       // Overflow Flag
#define DSPMINVAL                   0x433       // Minimum Value
#define DSPFFTORD                   0x434       // FFT Order
#define DSPFR                       0x435       // Framesize
#define DSFFTLEN                    0x436       // FFT Length
#define DSPMAXVAL                   0x437       //  Maximum Value
#define DSPRIFLAG                   0x438       // Real/Imaginary Flag
#define DSPCOUPLIST                 0x439       // Coupling List not used in some boards as Coupling List)
#define DSPIOMASK                   0x439       // Mask
#define DSPGAINLIST                 0x43A       // Gain List (not used in some boards)
#define DSPIOFFSET                  0x43A       // Offset
#define DSPFSMODE                   0x43B       // Sampling Rate Mode
#define DSPTRIGLEVEL                0x43C       // Trigger Level Value
#define DSPBUFLEN                   0x43D       // Buffer Length
#define DSPHOSTBUFNUM               0x43E       // Current Host Buffer Flag
#define DSPBUFNUM                   0x43F       // Current DSP Buffer Flag
#define DSPSCALEA                   0x440       // Data Scale Factor A
#define DSPOFFSETA                  0x441       // Dat Offset A
#define DSPWINSCL                   0x442       // Window Scaling Factor
#define DSPPHZREQ                   0x443       // Phase Data Required
#define DSPDUPFLG                   0x444       // Trace Duplicate Flag
#define DSPFILTADDR1                0x445       // Filter 1 Coefficient Address
#define DSPCHAN                     0x446       // Analog I/O Start Channel
#define DSPTRIGCHAN                 0x447       // Trigger Channel
#define DSPSCALEB                   0x448       // Data Scale Factor B
#define DSPOFFSETB                  0x449       // Data Offset B
#define DSPFILTADDR2                0x44A       // Filter 2 Coefficient Address
#define DSPMAXVALREA                0x44B       // Real Data Maximum
#define DSPMAXVALIMA                0x44C       // Imaginary Data Maximum
#define DSPFILTLEN1                 0x44D       // Filter 1 Length
#define DSPFILTLEN2                 0x44E       // Filter 2 Length
#define DSPLOGFLGA                  0x44F       // Log Flag A
#define DSPLOGFLGB                  0x450       // Log Flag B
#define DSPFILTTYPEA                0x451       // Filter Type A
#define DSPFSVALUE                  0x452       // Sampling Rate Value
#define DSPPWRFLGA                  0x453       // Power Spectra Flag A
#define DSPPWRFLGB                  0x454       // Power Spectra Flag B
#define DSPPWRCOEFFA                0x455       // Power Spectra Coefficient A
#define DSPPWRCOEFFB                0x456       // Power Spectra Coefficient B
#define DSPXFERFLGA                 0x457       // Transfer Function Flag A
#define DSPXFERFLGB                 0x458       // Transfer Function Flag B
#define DSPMAXVALA                  0x459       // Maximum Amplitude A
#define DSPMAXVALB                  0x45A       // Maximum Amplitude B
#define DSPNUMCHAN                  0x45B       // Number of Analog Channels
#define DSPBOARDSUBTYPE             0x45C       // Board Sub Type
#define DSPTIMDATAADDR              0x45D       // Time Data Base Address
#define DSPRIDATAADDR               0x45E       // Real/Imaginary Data Base Address
#define DSPMAGDATAADDR              0x45F       // Mgintude/Phase Data Base Address
#define DSPWNDDATAADDR              0x460       // Window Data Base Address
#define DSPMONITORFLG               0x461       // Monitor Mode
#define DSPIOMODULE                 0x462       // Module Indicator
```

```
#define DSPSERIALCTRL       0x463   // AIC Control Word/ CODEC Control Word 1
#define DSPLOGCOEFFA        0x464   // Log Coefficient 1
#define DSPLOGCOEFFB        0x465   // Log Coefficient 2
#define DSPADCOUNT          0x466   // Acquistion Delay Counter
#define DSPDACOUNT          0x467   // Stimulas Delay Counter
#define DSPSTMDATAADDR      0x468   // Stimulas Data Base Address
#define DSPSTMBUFLEN        0x469   // Stimulas Data Buffer Length
#define DSPFILTTYPEB        0x46A   // Filter Type B
#define DSPFILTQUANT        0x46B   // Filter Quantization
#define DSPFILTUPDFLG       0x46C   // Filter Coefficient Update Flag
#define DSPCODECCTRL1       0x46D   // CODEC Control Word 2
#define DSPCODECCTRL2       0x46E   // CODEC Control Word 3
```

Appendix A: Glossary

A/D	Analog-to-digital, as in A/D converter.
CODEC	COder - DECoder, used to describe a monolithic A/D and D/A device.
D/A	Digital-to-analog, as in D/A converter.
DOS	Disk-based operating system. Standard for IBM-compatible personal computers until advent of Windows.
DMA	Direct memory access.
DSP	Digital Signal Processing.
hex	Hexadecimal, or base-16 number system.
MOPS	Million operations per second.
MFLOPS	Million floating-point operations per second.
SRAM	Static random access memory.


QW8AF DIAGRAM
MEMORY MAP
INI RAM 1
CTRL STAT
CTRL & STATUS @ 0xE00003
LS 16 BITS READ_FIFO
ACCESS QWbus FS = 1 SLAVE PERIPHERALS
LS 16 BITS QWbus MASTER
ACCESS QWbus FS = 0 WRITE OTHER FIFOS READ.. INTERRUPT
LS 16 BITS QWbus COMMS
INT RAM 0
INT RAM 2
EXT 128K WORDS
EXT 32K WORDS
0x3 0x2 0x1 0x0
BYTE ORDERING
0 WS
B
1
0
A
0 WAIT STATE
QWbus INTERFACE
QWbus ADDRESS
ADDRESS CTRL 16 BIT DATA
RESET FIFO
CONTROL REGISTER
EXTERNAL MEMORY
32K=32 OR 128K=32
DSP32C PROCESSOR
HOST PORT
SERIAL PORT
HOST PORT INTERFACE
CTRL DATA ADDRESS
PC AT BUS
A4,5,6
D0..D31
D0..D15
D0..A11
D24..D30
SCLK
SRNC
RESET & CALIBRATE
ANALOG INPUT & OUTPUT CHANNELS
CHANNEL A
CHANNEL B
CHANNEL C
CHANNEL D
P1
LK5
VCC
SW1 MOST SIGNIFICANT BASE ADDRESS
SW2 LEAST SIGNIFICANT BASE ADDRESS AND QWbus ADDRESS SELECT
LK1 & LK2 DMA CHANNEL SELECT
LK3 PC INTERRUPT SELECT
LK4
FEMALE P2

Appendix A: Glossary

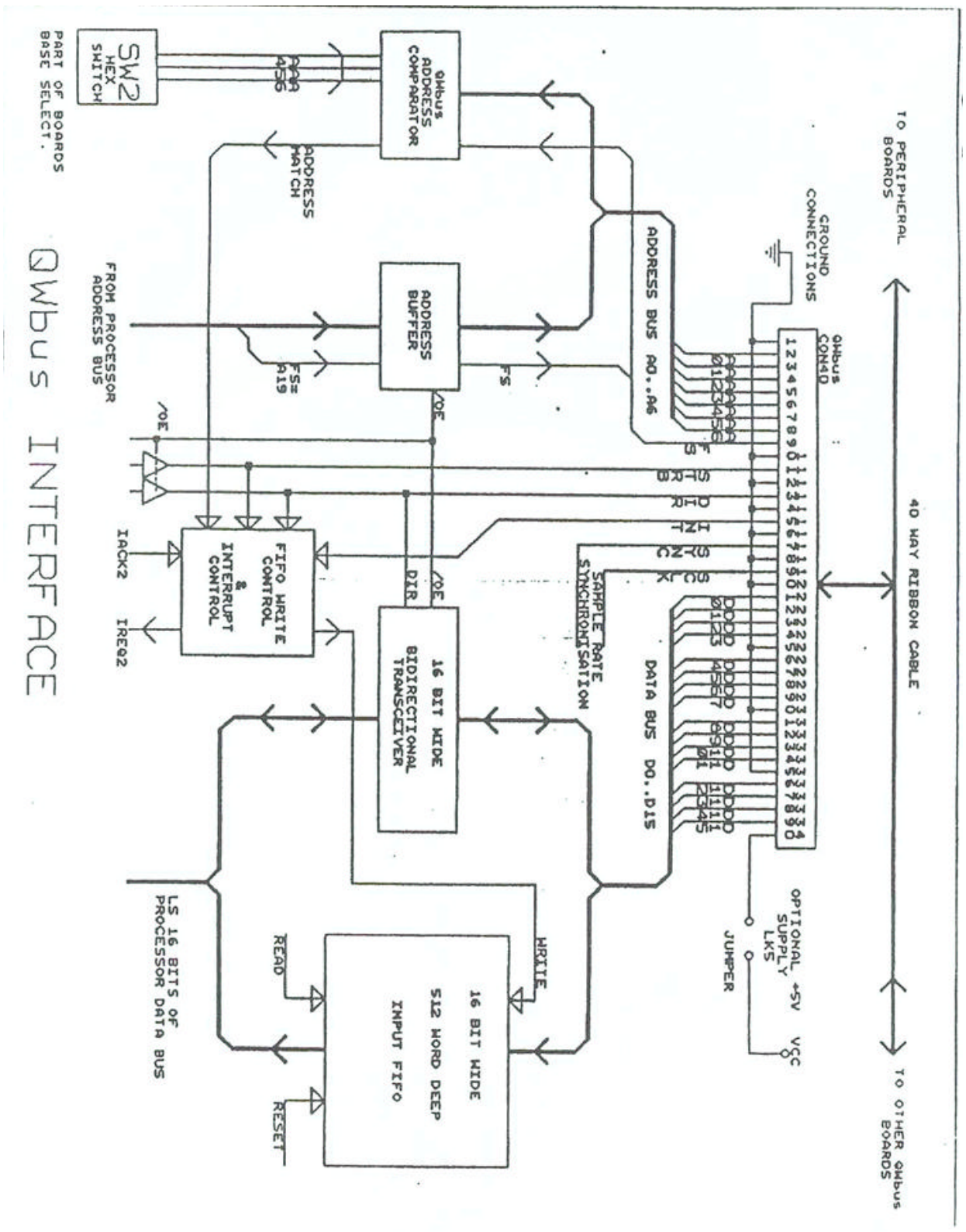
his page intntionally left blank

Appendix B: Block-Diagram Schematics

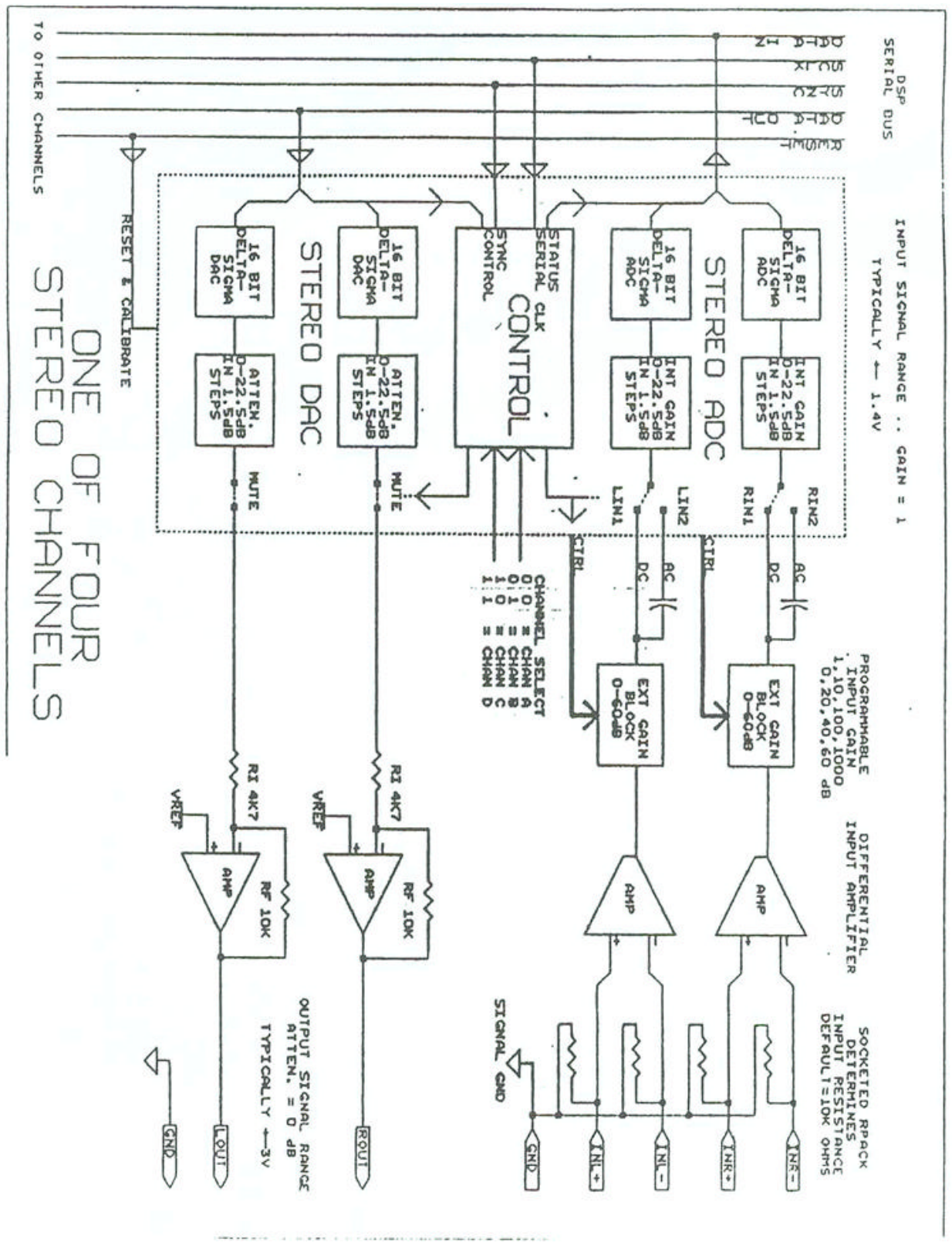
Appendix B:
Block-Diagram

this page intentionally left blank

Appendix C: CS4216 Data Sheets


QWbus INTERFACE
TO PERIPHERAL BOARDS
40 WAY RIBBON CABLE
TO OTHER QWbus BOARDS
PART OF BOARDS BASE SELECT.
SW2 HEX SWITCH
QWbus ADDRESS COMPARATOR
ADDRESS MATCH
GROUND CONNECTIONS
QWbus CON40
ADDRESS BUS A0..A6
FROM PROCESSOR ADDRESS BUS
ADDRESS BUFFER
FS
FS= A19
/OE
DIR
IACK2
IREQ2
FIFO WRITE CONTROL & INTERRUPT CONTROL
16 BIT WIDE BIDIRECTIONAL TRANSCEIVER
SAMPLE RATE SYNCHRONISATION
DATA BUS D0..D15
OPTIONAL +5V SUPPLY LKS
VCC
JUMPER
WRITE
READ
RESET
16 BIT WIDE
512 WORD DEEP
INPUT FIFO
LS 16 BITS OF PROCESSOR DATA BUS

Appendix C: CS4216
Data Sheets



OUTPUT DATA BIT DEFINITIONS

Sub-frame Bits 1 to 16

Left ADC Audio Data, MSB first, 2's complement coded.

Sub-frame Bits 17 to 24

17	18	19	20	21	22	23	24
RESERVED				0	ADV	LCL	RCL

ADV ADC Valid data bit.
0 - Invalid ADC data
1 - Valid ADC data
Indicates ADC has completed initialization after power-up, low power mode, or mute.

LCL Left ADC clipping indicator
0 - Normal
1 - Clipping

RCL Right ADC clipping indicator
0 - Normal
1 - Clipping

RESERVED bits can be 0 or 1

Sub-frame Bits 25 to 32

25	26	27	28	29	30	31	32
ER3	ER2	ER1	ER0	Ver3	Ver2	Ver1	Ver0

ER3-ER0 Error Word
0000 - Normal – No errors.
0001 - Input Sub-frame Bit 21 is set. Control data will not be loaded
0010 - Sync Pulse is incorrect. Causes the analog output to mute.
0011 - SCLK is outside the allowable range. Analog output mutes.

Ver3-Ver0 CS4216 Version Number
0000 = "A" (see Appendix A)
0001 = "B", "C", . . . (This data sheet)

Sub-frame Bits 33 to 48

Right ADC Audio Data, MSB first, 2's complement coded.

Sub-frame Bits 49 to 60

These bits are reserved, and can be 0 or 1.

Sub-frame Bits 61 to 64

61	62	63	64
DI1	DI2	DI3	DI4

DI1-DI4 These bits follow the state of the Digital Input pins. In SM3 DI3 and DI4 are used and unavailable. In SM4 DI2, DI3, & DI4 are not available as input bits.

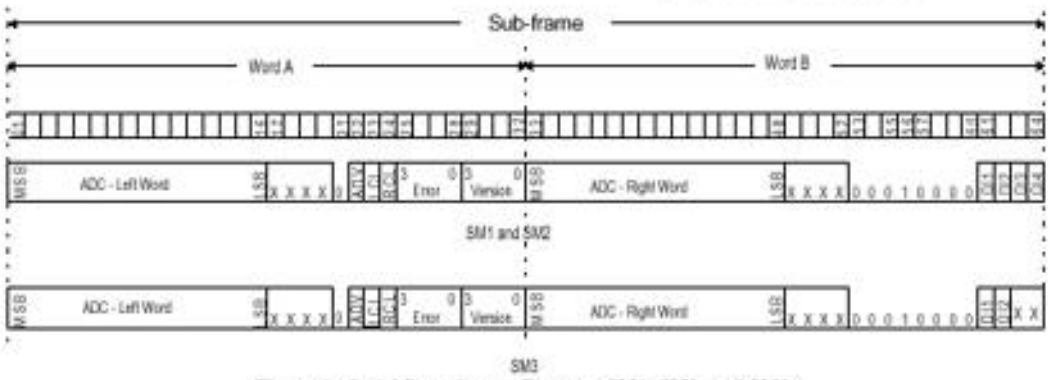


Figure 6. Serial Data Output Format - SM1, SM2, and SM3.

Appendix C: CS4216
Data Sheets

RECOMMENDED OPERATING CONDITIONS (AGND, DGND = 0V, all voltages with respect to 0V.)

Parameter		Symbol	Min	Typ	Max	Units
Power Supplies:	Digital	VD	4.75	5.0	5.25	V
	Analog	VA	4.75	5.0	5.25	V
Operating Ambient Temperature		TA	0	25	70	°C

ANALOG CHARACTERISTICS (T_A = 25°C; VA, VD = +5V; Input Levels: Logic 0 = 0V, Logic 1 = VD; 1 kHz Input Sine Wave; CLKIN = 24.576 MHz; SM1; Conversion Rate = 48 kHz; SCLK = 12.288 MHz; Measurement Bandwidth is 10 Hz to 20 kHz; Unless otherwise specified.)

Parameter *		Symbol	Min	Typ	Max	Units
Analog Input Characteristics - Minimum gain setting (0 dB); unless otherwise specified.						
ADC Resolution			16	-	-	Bits
ADC Differential Nonlinearity	(Note 1)		-	-	±0.9	LSB
Instantaneous Dynamic Range		IDR	80	85	-	dB
Total Harmonic Distortion		THD	-	-	0.01	%
Interchannel Isolation			-	80	-	dB
Interchannel Gain Mismatch			-	-	±0.5	dB
Frequency Response	(Note 1)		-0.5	-	+0.2	dB
Programmable Input Gain Span			21	22.5	24	dB
Gain Step Size			-	1.5	-	dB
Absolute Gain Step Error			-	-	0.75	dB
Gain Drift			-	100	-	ppm/°C
Offset Error	DC Coupled Inputs		-	±10	±100	LSB
	AC Coupled Inputs		-	±150	±400	LSB
Full Scale Input Voltage			2.5	2.8	3.1	V_{pp}
Input Resistance	(Notes 1,2)		20	-	-	kΩ
Input Capacitance	(Note 1)		-	-	15	pF

Notes: 1. This specification is guaranteed by characterization, not production testing.
2. Input resistance is for the input selected. Non-selected inputs have a very high (>1MΩ) input resistance.

* Parameter definitions are given at the end of this data sheet.

Mwave™ is a trademark of the IBM Corporation.

Specifications are subject to change without notice.

ANALOG CHARACTERISTICS (Continued)

Parameter *		Symbol	Min	Typ	Max	Units
Analog Output Characteristics - Minimum Attenuation; Unless Otherwise Specified.						
DAC Resolution			16	-	-	Bits
DAC Differential Nonlinearity	(Note 1)		-	-	±0.9	LSB
Total Dynamic Range		TDR	-	93	-	dB
Instantaneous Dynamic Range		IDR	80	83	-	dB
Total Harmonic Distortion	(Note 4)	THD	-	-	0.02	%
Interchannel Isolation	(Note 4)		-	80	-	dB
Interchannel Gain Mismatch			-	-	±0.5	dB
Frequency Response	(Note 1)		-0.5	-	+0.2	dB
Programmable Output Attenuation Span	(Note 3)		-45	-46.5	-	dB
Attenuation Step Size	(Note 3)		-	1.5	-	dB
Absolute Attenuation Step Error	(Note 3)		-	-	0.75	dB
Gain Drift			-	100	-	ppm/°C
REFBUF Output Voltage Maximum output current= 400 µA	(Note 5)		1.9	2.2	2.5	V
Offset Voltage			-	10	-	mV
Full Scale Output Voltage	(Note 4)		2.5	2.8	3.1	V_{pp}
Deviation from Linear Phase	(Note 1)		-	-	1	Degree
Out of Band Energy	(22 kHz to 100 kHz)		-	-60	-	dB
Power Supply						
Power Supply Current (Note 6)	Operating Power Down		- -	80 -	100 1	mA mA
Power Supply Rejection	(1 kHz)		-	40	-	dB

Notes: 3. Tested in SM3, Slave sub-mode, 128 BPF.
4. 10 kΩ, 100 pF load.
5. REFBUF load current must be DC. To drive dynamic loads, REFBUF must be buffered. AC variations in REFBUF current may degrade ADC and DAC performance.
6. Typically current: VA = 30mA, VD = 50mA. Power supply current does not include output loading.

* Parameter definitions are given at the end of this data sheet.

Appendix C: CS4216
Data Sheets

A/D Decimation Filter Characteristics

Parameter	Symbol	Min	Typ	Max	Units
Passband (Fs is conversion freq.)		0	-	0.45Fs	Hz
Frequency Response		-0.5	-	+0.2	dB
Passband Ripple		-	-	±0.2	dB
Transition Band		0.45Fs	-	0.55Fs	Hz
Stop Band		≥ 0.55Fs	-	-	Hz
Stop Band Rejection		80	-	-	dB
Group Delay		-	16/Fs	-	s
Group Delay Variation vs. Frequency		-		0.0	µs

D/A Interpolation Filter Characteristics

Parameter	Symbol	Min	Typ	Max	Units
Passband (Fs is conversion freq.)		0	-	0.45Fs	Hz
Frequency Response		-0.5	-	+0.2	dB
Passband Ripple		-	-	±0.1	dB
Transition Band		0.45Fs	-	0.55Fs	Hz
Stop Band		≥ 0.55Fs	-	-	Hz
Stop Band Rejection		74	-	-	dB
Group Delay		-	16/Fs	-	s
Group Delay Variation vs. Frequency		-	-	0.1/Fs	µs

ABSOLUTE MAXIMUM RATINGS (AGND, DGND = 0V, all voltages with respect to 0V.)

Parameter	Symbol	Min	Typ	Max	Units
Power Supplies: Digital	VD	-0.3	-	6.0	V
Analog	VA	-0.3	-	6.0	V
Input Current (Except Supply Pins)		-	-	±10.0	mA
Analog Input Voltage		-0.3	-	VA+0.3	V
Digital Input Voltage		-0.3	-	VD+0.3	V
Ambient Temperature (Power Applied)		-55	-	+125	°C
Storage Temperature		-65	-	+150	°C

Warning: Operation beyond these limits may result in permanent damage to the device.
Normal operation is not guaranteed at these extremes.

INPUT DATA BIT DEFINITIONS

Sub-frame bits 1 to 16

Left DAC Audio Data, MSB first, 2's complement coded.

Sub-frame Bits 17 to 24

17	18	19	20	21	22	23	24
0	0	0	0	EXP	MUTE	ISL	ISR

EXP Expand bit
Reserved. Must be set to zero.

MUTE Mute D/A Outputs
0 - Normal Outputs
1 - Mute Outputs

ISL Select Left Input Mux
0 - Select LIN1
1 - Select LIN2

ISR Select Right Input Mux
0 - Select RIN1
1 - Select RIN2

Sub-frame Bits 25 to 32

25	26	27	28	29	30	31	32
LG3	LG2	LG1	LG0	RG3	RG2	RG1	RG0

LG3-LG0 Sets left input gain.
LG3 is the MSB. LG0 represents 1.5 dB.
0000 = no gain.
1111 = +22.5 dB gain

RG3-RG0 Sets right input gain.
RG3 is the MSB. RGO represents 1.5 dB.
0000 = no gain

Sub-frame Bits 33 to 48

Right DAC audio data MSB first, 2's complement coded.

Sub-frame Bits 49 to 50

Must be zero.

Sub-frame Bits 51 to 60

	51	52	53	54	55	56	57	58	59	60
*	LA4	LA3	LA2	LA1	LA0	RA4	RA3	RA2	RA1	RA0
†	0	0	LA3	LA2	LA1	LA0	RA3	RA2	RA1	RA0

LA4-LA0 Sets left output attenuation

†SM1, 2	*SM3,4
LA3 is the MSB.	LA4 is the MSB.
0000 = no attenuation	00000 = no attenuation
1111 = -22.5 dB	11111 = -46.5 dB
LA0 represents 1.5 dB.	

RA4-RA0 Sets right output attenuation

†SM1, 2	*SM3,4
RA3 is the MSB.	RA4 is the MSB.
0000 = no attenuation	00000 = no attenuation
1111 = -22.5 dB	11111 = -46.5 dB
RA0 represents 1.5 dB.	

Sub-frame Bits 61 to 64

61	62	63	64
DO1	DO2	DO3	DO4

DO1-DO4 Set the logic level on the 4 digital output pins. In SM3 DO3 and DO4 are not available. In SM4 DO2, DO3, & DO4 are not available.

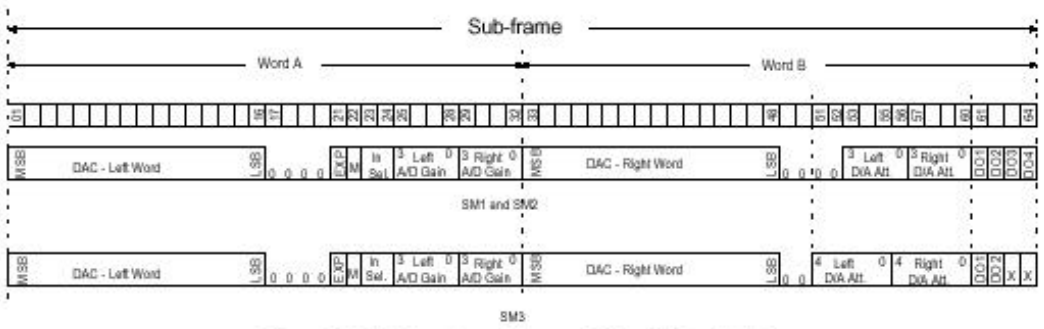


Figure 5. Serial Data Input Format - SM1, SM2, and SM3.

Appendix C: CS4216
Data Sheets

Appendix C: CS4216
Data Sheets

this page intentionally left blank